



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

**CORRELATION IMMUNITY, AVALANCHE FEATURES, AND
OTHER CRYPTOGRAPHIC PROPERTIES OF GENERALIZED
BOOLEAN FUNCTIONS**

by

Thor Martinsen

September 2017

Dissertation Supervisor:

Pantelimon Stănică

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE September 2017		3. REPORT TYPE AND DATES COVERED Dissertation
4. TITLE AND SUBTITLE CORRELATION IMMUNITY, AVALANCHE FEATURES, AND OTHER CRYPTOGRAPHIC PROPERTIES OF GENERALIZED BOOLEAN FUNCTIONS			5. FUNDING NUMBERS	
6. AUTHOR(S) Thor Martinsen				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) This dissertation investigates correlation immunity, avalanche features, and the bent cryptographic properties for generalized Boolean functions defined on \mathbb{V}_n with values in \mathbb{Z}_q . We extend the concept of correlation immunity from the Boolean case to the generalized setting, and provide multiple construction methods for order 1 and higher correlation immune generalized Boolean functions. We establish necessary and sufficient conditions for generalized Boolean functions. Additionally, we discuss correlation immune and rotation symmetric generalized Boolean functions, introducing a construction method along the way. Using a graph-theoretic and probabilistic frame of reference, we subsequently establish several, increasingly stringent, strict avalanche criteria along with a construction method for generalized Boolean functions. We introduce the notion of a uniform avalanche criterion and demonstrate that generalized Boolean functions that satisfy this criterion are also order 1 correlation immune and always have Boolean function components that are both order 1 correlation immune and satisfy the strict avalanche criterion. We subsequently investigate linear structures, directional derivatives and define a unit vector gradient for generalized Boolean function. We introduce the Walsh-Hadamard transform of a generalized Boolean function along with the notion of generalized bent Boolean functions. We provide a construction of generalized bent Boolean functions with outputs in \mathbb{Z}_8 and establish necessary conditions for generalized bent Boolean functions.				
14. SUBJECT TERMS Cryptography, coding theory, Boolean functions, generalized Boolean functions, correlation immunity, strict avalanche criterion, bent functions, cyber, information warfare, information security, communications security.			15. NUMBER OF PAGES 161	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**CORRELATION IMMUNITY, AVALANCHE FEATURES, AND OTHER
CRYPTOGRAPHIC PROPERTIES OF GENERALIZED BOOLEAN FUNCTIONS**

Thor Martinsen
Commander, United States Navy
B.S., Thomas Edison State College, 1996
B.A., Skidmore College, 2007
M.S., Naval Postgraduate School, 2007

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN APPLIED MATHEMATICS

from the

**NAVAL POSTGRADUATE SCHOOL
September 2017**

Author: Thor Martinsen

Approved by: Pantelimon Stănică
Professor of Applied Mathematics
Dissertation Supervisor

Craig W. Rasmussen
Professor of Applied Mathematics

David R. Canright
Associate Professor of Applied
Mathematics

George W. Dinolt
Professor of Practice in Cyber
Operations

Sugata Gangopadhyay
Associate Professor of Comp. Sci. and
Eng., Indian Inst. of Techn.-Roorkee

Approved by: Craig W. Rasmussen
Chair, Department of Applied Mathematics

Approved by: Douglas Moses
Vice Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This dissertation investigates correlation immunity, avalanche features, and the bent cryptographic properties for generalized Boolean functions defined on \mathbb{V}_n with values in \mathbb{Z}_q . We extend the concept of correlation immunity from the Boolean case to the generalized setting, and provide multiple construction methods for order 1 and higher correlation immune generalized Boolean functions. We establish necessary and sufficient conditions for generalized Boolean functions. Additionally, we discuss correlation immune and rotation symmetric generalized Boolean functions, introducing a construction method along the way. Using a graph-theoretic and probabilistic frame of reference, we subsequently establish several, increasingly stringent, strict avalanche criteria along with a construction method for generalized Boolean functions. We introduce the notion of a uniform avalanche criterion and demonstrate that generalized Boolean functions that satisfy this criterion are also order 1 correlation immune and always have Boolean function components that are both order 1 correlation immune and satisfy the strict avalanche criterion. We subsequently investigate linear structures, directional derivatives and define a unit vector gradient for generalized Boolean function. We introduce the Walsh-Hadamard transform of a generalized Boolean function along with the notion of generalized bent Boolean functions. We provide a construction of generalized bent Boolean functions with outputs in \mathbb{Z}_8 and establish necessary conditions for generalized bent Boolean functions.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	Background	1
1.2	Contributions	2
1.3	Dissertation Organization	3
2	Basic Properties of Generalized Boolean Functions	5
2.1	Preliminaries	5
2.2	The Algebraic Normal Form for Generalized Boolean Functions	7
2.3	Fourier Transforms and Generalized Boolean Functions	11
2.4	Balance and Symmetry	13
3	Correlation Immune Generalized Boolean Functions	19
3.1	Introduction	19
3.2	Correlation Immune Constructions	23
3.3	A Higher Order Correlation Immune Construction	27
3.4	New From Old Correlation Immune Generalized Boolean Functions	43
3.5	Necessary and Sufficient Conditions for Correlation Immune Generalized Boolean Functions.	46
3.6	Correlation Immunity and the Walsh-Hadamard Transform.	52
3.7	Rotation Symmetric Correlation Immune Generalized Boolean Functions	54
4	Avalanche Criteria for Generalized Boolean Functions	67
4.1	Introduction	67
4.2	A Strict Avalanche Criterion Construction for Boolean Functions	68
4.3	A Probabilistic Strict Avalanche Criterion	77
4.4	Global and Uniform Avalanche Criteria	84
4.5	Necessary and Sufficient Conditions for a Generalized Strict Avalanche Criterion	90
4.6	The Connection between the Uniform Avalanche Criterion and Correlation Immunity	96

4.7	Linear Structures and the Globally Uniform Gradient	102
5	Generalized Bent Boolean Functions	107
5.1	Introduction	107
5.2	Generalized Bent Boolean Functions	108
5.3	Construction of Generalized Bent Functions in \mathcal{GB}_n^8	110
5.4	Necessary Conditions for Generalized Bent Functions.	112
6	Conclusion and Future Research	117
6.1	Conclusion.	117
6.2	Future Research.	118
APPENDIX A	Table of Nontrivial Binomial Bisections	119
APPENDIX B	Binomial Bisection Program	121
APPENDIX C	Some Linear Orthogonal Arrays for Higher Order Correlation Immune Generalized Boolean Function Constructions	127
LIST OF REFERENCES		137
INITIAL DISTRIBUTION LIST		141

List of Figures

Figure 3.1	q -ary sequence generator	47
Figure 4.1	S-box using generalized Boolean function pointers	90
Figure 4.2	Labeled hypercube corresponding to the generalized Boolean function in Table 4.9	101

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 3.1	Conditional probability table for a Boolean function	20
Table 3.2	Conditional probability table for an order 1 correlation immune Boolean function	21
Table 3.3	A $CI(1)$ Boolean function $f \in \mathcal{B}_4$	24
Table 3.4	A $CI(1)$ generalized Boolean function $f \in \mathcal{GB}_4^4$	27
Table 3.5	A $CI(2)$ generalized Boolean function $f \in \mathcal{GB}_5^4$	37
Table 3.6	Some orthogonal arrays and associated generalized Boolean function parameters	43
Table 3.7	A Siegenthaler constructed $CI(1)$ function $f \in \mathcal{GB}_4^4$	45
Table 3.8	A correlation immune generalized Boolean function construction failure	46
Table 3.9	A <i>non</i> – $CI(1)$ function $f \in \mathcal{GB}_4^4$, where $H_f(\mathbf{w}) = 0$	54
Table 3.10	A <i>RotS</i> and $CI(1)$ generalized Boolean function $f \in \mathcal{GB}_4^4$	57
Table 4.1	A <i>SAC</i> and $CI(1)$ Boolean function $f \in \mathcal{B}_4$	77
Table 4.2	A <i>PSAC</i> generalized Boolean function $f \in \mathcal{GB}_4^4$	83
Table 4.3	Vertex invariance probability for a <i>SAC</i> Boolean function	85
Table 4.4	Vertex invariance probability for a globally <i>SAC</i> Boolean function	86
Table 4.5	A <i>UAC</i> generalized Boolean function $f \in \mathcal{GB}_4^4$	89
Table 4.6	A <i>non</i> – <i>UAC</i> $CI(1)$ generalized Boolean function $f \in \mathcal{GB}_4^4$	97
Table 4.7	<i>UAC</i> function $f_1 \in \mathcal{GB}_4^4$	99
Table 4.8	<i>UAC</i> function $f_2 \in \mathcal{GB}_4^4$	99
Table 4.9	A <i>PSAC</i> and 1-resilient generalized Boolean function $f_1 f_2 = f \in \mathcal{GB}_5^4$	100

Table 4.10	A UAC function $f \in \mathcal{GB}_4^4$, without e_i as a linear structure	103
Table 4.11	Gradients for two UAC generalized Boolean functions f_1 and f_2	106
Table A.1	Nontrivial binomial bisections	120

List of Acronyms and Abbreviations

ANF	Algebraic normal form
CI	Correlation immune
CI(<i>t</i>)	Correlation immune of order <i>t</i>
gbent	Generalized bent Boolean function
gCI	Generalized correlation immune
GAC	Global avalanche criterion
GSAC	Generalized strict avalanche criterion
LFSR	Linear feedback shift register
NPS	Naval Postgraduate School
PSAC	Probabilistic strict avalanche criterion
RotS	Rotation symmetric
SAC	Strict avalanche criterion
S-box	Substitution box
UAC	Uniform avalanche criterion
XOR	Exclusive OR

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

The Nation that makes a great
distinction between its scholars and its
warriors will have its thinking done by
cowards and its fighting done by fools.

Thucydides✎

This dissertation investigates cryptographic properties of generalized Boolean functions. Generalized Boolean functions, $f : \mathbb{V}_n \rightarrow \mathbb{Z}_q$, are functions from the vector space of binary vectors of length n to a ring of integers modulo q . The classical Boolean case, where $q = 2$, has been studied extensively. Such Boolean functions are frequently used as components in cryptographic algorithms. Much less is currently known about the generalized case, where $q > 2$. From a cryptologist's point of view, generalized Boolean functions show promise in a number of cryptographic applications, including those in the quantum environment.

In this dissertation we investigate correlation immunity, avalanche features, and the bent property of generalized Boolean functions. We extend the concept of correlation immunity to the generalized setting and establish several new results for correlation immune generalized Boolean functions. We present several algorithms for the construction of order 1, higher order, concatenated, and rotation symmetric correlation immune generalized Boolean functions. We also establish necessary and sufficient conditions for correlation immune generalized Boolean functions. Doing so is important because generalized Boolean functions suitable for cryptographic applications must not only be correlation immune, but all of their constituent Boolean function components must also be correlation immune. Using a graph-theoretic and probabilistic frame of reference, we then investigate avalanche features of generalized Boolean functions. We establish several, increasingly stringent, avalanche criteria for generalized Boolean functions. This line of investigation culminates in the development of the uniform avalanche criterion (UAC). We demonstrate that generalized Boolean functions that satisfy the UAC are also order 1 correlation immune and contain Boolean function components all of which are order 1 correlation immune and satisfy the strict avalanche criterion (SAC). We investigate linear structures and directional deriva-

tives of UAC compliant generalized Boolean functions. We also introduce and demonstrate the utility of the concept of a uniform generalized Boolean function unit vector gradient. Finally, we present a selection of results on generalized bent Boolean functions taken from the dissertation author's previously published papers on the topic. In particular, we introduce the Walsh-Hadamard transform of generalized Boolean functions, and define the concept of a generalized bent Boolean function. We subsequently provide a construction of generalized bent Boolean functions with outputs in \mathbb{Z}_8 , and establish necessary conditions for generalized bent Boolean functions.

Acknowledgments

Non sibi sed patriae ✎

I rightfully begin by thanking my dissertation committee members, first and foremost among whom is Professor Pante Stănică, my dissertation supervisor. His academic tutelage, patience, and unwavering support throughout my studies have left me with a debt I fear I will struggle to repay. The famous Norwegian mathematician Niels Henrik Abel once said, “It appears to me that if one wishes to make progress in mathematics, one should study the masters and not the pupils.” I am immensely grateful for the opportunity I have been given to study under Pante. I also wish to thank Professor Craig Rasmussen whose dedicated efforts prepared me well for the combinatorics portions of my written and oral qualifying exams. Throughout my time at NPS he has been a source of great encouragement and support. Thanks as well go to Professors David Canright and George Dinolt. Last but not least, I would like to thank Professor Sugata Gangopadhyay, who despite his geographic separation and significant teaching and research demands, agreed to serve as an external member of my committee. Although not on my committee, I would be remiss in my duties were I to fail to recognize Assistant Professor Jeremy Kozdon for his assistance in speeding up my computer code and helping me navigate the hazards of the Naval Postgraduate School’s high performance computer.

My military duties have resulted in an academic journey that is anything but typical. During the past two decades there have been many educators who have selflessly given of themselves, expanded my academic horizons, and in so doing have had a profound impact on my life. In addition to those already mentioned, I would like to thank Professor emeritus Hal Fredricksen, who always loved a good mathematical story. I hope the story told herein meets with his approval. I am also grateful for the outstanding mentorship I received from Professors Sarah Stebbins, Christopher Whann, and David Vella during my time at Skidmore College. A special thanks goes out to Professors emeriti David Cecil and Alan Kay from Texas A&M University Kingsville, who first introduced me to Abstract Algebra and Real Analysis and as such had a hand in setting me upon this path. I am also grateful to

Mrs. Ruth Lattimore from the University of Maryland, who instructed me at the onset of this journey. Pericles once said, “what you leave behind is not what is engraved in stone monuments, but what is woven into the lives of others.” While I may never be able to repay what I have been given, I hope that I may now begin to pay it forward as I assume my duties as a Navy Permanent Military Professor.

I remain immensely grateful to the United States Navy for the many opportunities the service has afforded me and for the trust it continues to place in my leadership abilities. I am also thankful to my fellow shipmates, both past and present. I have the privilege of serving in the great state of California due to the actions of Commodore John D. Sloat, who in July 1846 raised Old Glory above the Customs House, located not more than a mile from where I sit. I enjoy the necessary safety and peace of mind with which to carry out my daily academic duties due to Sailors, across the globe, who stand the watch. As Winston Churchill once said, “we sleep safely at night because rough men stand ready to visit violence on those who would harm us.”

I conclude by thanking those most important to me, namely my family. Thanks to my parents, Finn and Unni-lise, as well as my big sister Heidi. Most importantly, I would like to thank my wife, Dione. She is the love of my life and has been the most ardent supporter of all my academic and military endeavors throughout these many years. Thanks as well go to my son Corey. He is my best friend and one of the brightest and most fun people I know. Dee and Corey have sacrificed so much for me to be where I find myself today, both professionally and academically. Without their love and support this work would neither have been possible nor worthwhile. Thank you.

CHAPTER 1:

Introduction

He who loves practice without theory
is like the sailor who boards ship
without a rudder and compass and
never knows where he may cast.

Leonardo da Vinci ✍

1.1 Background

Functions $f : \mathbb{V}_n \rightarrow \mathbb{F}_2$ from the vector space \mathbb{V}_n of all binary vectors of length n , to the finite field of two elements are known as Boolean functions. These functions are essential components in modern cryptography and error correction codes. As such, they have been the subject of intense study for the past 50 years, and much is therefore known about them. In contrast, much less is understood about generalized Boolean functions from the vector space \mathbb{V}_n of all binary vectors of length n , to \mathbb{Z}_q , where $q \geq 2$. Yet, these functions also show great promise of utility in future information, communications, and defense technologies.

The goal of this research has been to increase our understanding of generalized Boolean functions which satisfy certain cryptographic properties. Specifically, generalized Boolean functions which are correlation immune or satisfy strict avalanche criteria. As our starting point, we use existing Boolean functions research and then attempt, where possible, to extend these results into the more general setting. Much of Boolean function research has a tendency to be highly theoretical; while some of this research inevitably will follow suit, we have, whenever possible, tried to supply the reader with a generous number of examples as well as a fair number of algorithms with which they can go about constructing the functions under consideration.

1.2 Contributions

This dissertation makes the following contributions to the study of generalized Boolean functions:

- We define the algebraic normal form (ANF) of a generalized Boolean function and demonstrate a method of deriving the ANF using the function's truth table.
- Given function parameters n and q , we provide respective counts for the number of balanced and symmetric generalized Boolean functions in n variables with output values in \mathbb{Z}_q .
- We present several theorems regarding nontrivial binomial bisections, and provide a complete list of all binomial bisection solutions for $n \leq 51$.
- We extend the concept of correlation immunity from the Boolean case to the generalized setting.
- We provide an algorithm with which to construct a large class of correlation immune (order 1) generalized Boolean functions.
- Using linear orthogonal arrays we demonstrate a method of creating higher order correlation immune generalized Boolean functions.
- We extend and prove a generalized version of the Siegenthaler correlation immune Boolean function construction method, whereby two correlation immune (order t) generalized Boolean functions in n variables are combined to create a correlation immune (order t) generalized Boolean function in $n + 1$ variables.
- We establish necessary and sufficient conditions which ensure that both a generalized Boolean function as well as its Boolean function components are all correlation immune.
- We investigate correlation immune and rotation symmetric generalized Boolean functions and introduce a construction method for such functions.
- We establish an upper bound for the number of rotation symmetric (*RotS*) generalized Boolean functions, and prove that there are no balanced and *RotS* generalized Boolean functions in p variables with output values in \mathbb{Z}_q , for odd prime p and $q > 2$.
- Using a graph-theoretic and probabilistic frame of reference, we establish several, strict avalanche criteria including the notion of a uniform avalanche criterion (UAC).
- We prove that generalized Boolean functions which satisfy the uniform avalanche criterion are also order 1 correlation immune.

- We prove that generalized Boolean functions which satisfy the uniform avalanche criterion have Boolean function components which are all both SAC and order 1 correlation immune.
- We investigate linear structures and directional derivatives of UAC-compliant generalized Boolean function and introduce the concept of a generalized Boolean function unit-vector gradient.
- We introduce the Walsh-Hadamard transform of generalized Boolean functions, and define perfect nonlinear generalized Boolean functions and generalized bent Boolean functions.
- We provide a construction of generalized bent Boolean functions in n variables with output values in \mathbb{Z}_8 .
- We further establish necessary conditions for generalized bent Boolean functions.

1.3 Dissertation Organization


This dissertation is divided into six chapters and three appendices. In addition to the introductory chapter in which you now find yourself, the remaining chapters are laid out as follows. Chapter 2 contains definitions and preliminary generalized Boolean function material. This is followed by Chapters 3–5, which contain the bulk of the dissertation research, including all major results. Chapter 3 deals with correlation immune generalized Boolean functions, whereas Chapter 4 tackles strict avalanche criteria. Chapter 5 contains a brief overview of the generalized bent property along with a selection of results taken from the this author’s previously published papers on this topic. Chapter 6 includes the dissertation conclusion along with a short discussion of follow-on research possibilities. This is followed by three appendices, the two first of which include a list of nontrivial binomial bisections along with the *Julia* parallel computer search program which generated the results. The final appendix includes a list of a few linear orthogonal arrays suitable for construction of higher order correlation immune generalized Boolean functions.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:

Basic Properties of Generalized Boolean Functions

Sic Parvis Magna

Sir Francis Drake 

In this chapter we begin by covering some basic definitions and properties which we will make use of throughout this dissertation.

2.1 Preliminaries

In a similar manner to what was done in [44], we will throughout this dissertation use the following definitions: We denote the set of integers, real numbers and complex numbers by \mathbb{Z} , \mathbb{R} and \mathbb{C} , respectively. We further denote the ring of integers modulo q by \mathbb{Z}_q . The vector space \mathbb{V}_n , sometimes alternatively referred to as \mathbb{F}_2^n , is the space of all n -tuples $\mathbf{x} = (x_n, \dots, x_1)$ of elements from \mathbb{F}_2 with the standard operations. By “+” we denote addition over \mathbb{Z} , \mathbb{R} and \mathbb{C} , whereas “ \oplus ” denotes addition over \mathbb{V}_n for all $n \geq 1$. Addition modulo q is denoted by “+” and it is understood from the context. If $\mathbf{x} = (x_n, \dots, x_1)$ and $\mathbf{y} = (y_n, \dots, y_1)$ are in \mathbb{V}_n , we define the scalar (or inner) product by $\mathbf{x} \cdot \mathbf{y} = x_n y_n \oplus \dots \oplus x_2 y_2 \oplus x_1 y_1$. The cardinality of the set S is denoted by $|S|$, and the conjugate of a bit b will be denoted by \bar{b} . If $z = a + b\iota \in \mathbb{C}$, then $|z| = \sqrt{a^2 + b^2}$ denotes the absolute value of z , and $\bar{z} = a - b\iota$ denotes the complex conjugate of z , where $\iota^2 = -1$, and $a, b \in \mathbb{R}$. The concatenation of two vectors \mathbf{x} and \mathbf{y} is denoted $\mathbf{x} \parallel \mathbf{y}$. Additionally, as in [11], we use the Landau symbol \mathcal{O} with its usual meaning, that is, $F = \mathcal{O}(G)$ means $|F(x)| \leq c|G(x)|$ holds for some positive constant c , and x sufficiently large.

Definition 2.1. A function from \mathbb{V}_n to \mathbb{F}_2 is called a Boolean function. The algebra of all Boolean functions on \mathbb{V}_n is denoted by \mathcal{B}_n [11].

Definition 2.2. We call a function from \mathbb{V}_n to \mathbb{Z}_q , where q is a positive integer such that $q \geq 2$, a *generalized Boolean function* on n variables [42]. We denote the set of such functions by \mathcal{GB}_n^q . If $q = 2$, we obtain the previously defined classical Boolean functions [44].

For a given n , there are a total of 2^n possible Boolean input vectors, each of which can in turn be mapped to q possible outputs. Therefore the total number of Boolean functions is $|\mathcal{B}_n| = 2^{2^n}$, whereas the total number of generalized Boolean functions is $|\mathcal{GB}_n^q| = q^{2^n}$. Given the fact that these formulae are double-exponential, the number of possible functions quickly becomes astronomical even for input vectors of relatively modest dimensions. For example, given input vectors of size $n = 7$ and output values in \mathbb{Z}_5 , the number of generalized Boolean functions is $|\mathcal{GB}_7^5| \approx 2.94 \times 10^{89}$. By comparison, the number of atoms in the observable universe is estimated to be between 10^{78} and 10^{82} .

As was done in [44], for any function $f \in \mathcal{GB}_n^q$ and $2^{k-1} < q \leq 2^k$, we associate a unique sequence of Boolean functions $a_i \in \mathcal{B}_n$ ($i = 0, 1, \dots, k-1$) such that

$$f(\mathbf{x}) = a_0(\mathbf{x}) + 2a_1(\mathbf{x}) + \dots + 2^{k-1}a_{k-1}(\mathbf{x}), \text{ for all } \mathbf{x} \in \mathbb{V}_n. \quad (2.1)$$

Definition 2.3. A *generalized Boolean function* $f(\mathbf{x})$ in n variables is a map from \mathbb{V}_n to \mathbb{Z}_q . In a manner similar to that in [11], the q -ary sequence defined by $(f(\mathbf{v}_0), f(\mathbf{v}_1), \dots, f(\mathbf{v}_{2^n-1}))$, where $\mathbf{v}_0 = (0, \dots, 0, 0), \mathbf{v}_1 = (0, \dots, 0, 1), \dots, \mathbf{v}_{2^n-1} = (1, \dots, 1, 1)$ is denoted by f and is called the *truth table* of $f(\mathbf{x})$.

Definition 2.4. The *Hamming weight* of a vector $\mathbf{x} = x_1 \cdots x_n$ (often written as $\mathbf{x} = (x_1, \dots, x_n)$), denoted by $wt(\mathbf{x})$, is the number of nonzero x_i , where $x_i \in \mathbb{Z}_q$ [26]. The Hamming weight of a function $f(\mathbf{x})$ is the Hamming weight of its truth table [11].

Definition 2.5. Given two q -ary vectors, \mathbf{x} and \mathbf{y} of length n , the *Hamming distance* between the two vectors, denoted $d(\mathbf{x}, \mathbf{y})$, is the number of indices where their values differ. Similarly, the Hamming distance between two n -variable functions $f(\mathbf{x})$ and $g(\mathbf{x})$, denoted $d(f, g)$ is defined as the number of indices for which their truth tables differ.

2.2 The Algebraic Normal Form for Generalized Boolean Functions

Definition 2.6. [11] Let $f \in \mathcal{B}_n$ be a Boolean function and let $\mathbf{i} = (i_1, \dots, i_n)$ and $\mathbf{x}^{\mathbf{i}} := x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. The Boolean function f is expressed in *Algebraic Normal Form* in the following manner:

$$f(\mathbf{x}) = \bigoplus_{i=0}^{2^n-1} c_i \cdot \mathbf{x}^{\mathbf{i}},$$

where $c_i \in \mathbb{F}_2$ and $\mathbf{i} \in \mathbb{V}_n$, is the lexicographically ordered binary expansion of index i .

Example 2.7. Consider the Boolean function $f(\mathbf{x}) = x_1 \oplus x_2 x_3 \oplus x_4$. Using the above definition it can be represented in ANF as:

$$f(\mathbf{x}) = 0 \cdot x_1^0 x_2^0 x_3^0 x_4^0 \oplus 1 \cdot x_1^1 x_2^0 x_3^0 x_4^0 \oplus \cdots \oplus 1 \cdot x_1^0 x_2^1 x_3^1 x_4^0 \oplus \cdots \oplus 1 \cdot x_1^0 x_2^0 x_3^0 x_4^1 \oplus \cdots \oplus 0 \cdot x_1^1 x_2^1 x_3^1 x_4^1.$$

Building upon this, we now define the Algebraic Normal Form for generalized Boolean functions as follows:

Definition 2.8. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function such that $f(\mathbf{x}) = a_0(\mathbf{x}) + 2a_1(\mathbf{x}) + \cdots + 2^{k-1}a_{k-1}(\mathbf{x})$, where $2^{k-1} < q \leq 2^k$. Let $\mathbf{j} = (j_1, \dots, j_n)$ and $\mathbf{x}^{\mathbf{j}} := x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n}$. We then define the *Algebraic Normal Form* of f in the following manner:

$$f(\mathbf{x}) = \sum_{i=0}^{k-1} 2^i \left(\bigoplus_{j=0}^{2^n-1} c_{ij} \mathbf{x}^{\mathbf{j}} \right),$$

where $c_j \in \mathbb{F}_2$, $\mathbf{j} \in \mathbb{V}_n$ is the lexicographically ordered binary expansion of index j , and the summation is carried out modulo 2^k .

It is relatively straightforward to recognize the existence and uniqueness of the ANF representation of generalized Boolean functions by considering the following: First, each vector,

$\mathbf{v} \in \mathbb{V}_n$, utilized in the ANF is unique and establishes a surjective map between \mathbb{V}_n and \mathcal{B}_n . Secondly, since $|\mathbb{V}_n| = 2^n$, the power set of \mathbb{V}_n has cardinality $|\mathcal{P}(\mathbb{V}_n)| = 2^{2^n} = |\mathcal{B}_n|$. Finally, the binary expansion of any integer q is unique. Given the ANF of a generalized Boolean function f , we can create the truth table of the function by simply using the ANF and evaluating, in turn, each of the 2^n lexicographically ordered input vectors. In order to proceed in the opposite direction and transform a truth table into an ANF expression, we first perform a binary expansion of each q -ary entry in the truth table, thereby creating multiple binary truth tables, one for each respective 2^k component of f , where $0 \leq k \leq \log_2 q$. Subsequently we perform the divide-and-conquer butterfly algorithm (see the description of Carlet in [7]) on each of the constituent binary truth tables and produce the corresponding 2^k -associated ANF components of the generalized Boolean function.

Example 2.9. Suppose we want to find the ANF for a function, $f \in \mathcal{GB}_3^4$, with the truth table $f = 02032012$. We begin by finding the binary truth tables a_0 and a_1 associated with 2^0 and 2^1 respectively by performing a binary expansion of f :

f	a_0	a_1
0	0	0
2	0	1
0	0	0
3	1	1
2	0	1
0	0	0
1	1	0
2	0	1

Having done so, we then apply the following algorithm to each of the binary truth tables.

Algorithm 1 TT to ANF (Butterfly algorithm) – see [7]

- 1: Write the truth-table of f , in which the binary vectors of length n are in lexicographic order.
 - 2: Let f_0 be the restriction of f to $\mathbb{F}_2^{n-1} \times \{0\}$ and f_1 the restriction of f to $\mathbb{F}_2^{n-1} \times \{1\}$. The truth-table of f_0 (resp. f_1) corresponds to the upper (resp. lower) half of the table of f ; replace the values of f_1 by those of $f_0 \oplus f_1$
 - 3: Apply recursively step 2, separately to the functions now obtained in the places of f_0 and f_1 .
 - 4: The algorithm terminates when it arrives at functions on one variable each. At this point the global table gives the values of the ANF of f .
-

For a_0 this yields the following.

a_0						ANF
0	f_0	0	f_0	0	f_0	0
0		0		0	f_1	0
0		0		0	f_0	0
1		1		1	f_1	1
0	f_1	0	f_0	0	f_0	0
0		0		0	f_1	0
1		1		1	f_0	1
0		1		1	f_1	0

Reading off the ANF column we recover the 2^0 -associated ANF-component of f :

$$\begin{aligned}
 a_0(\mathbf{x}) = & 0 \cdot x_1^0 x_2^0 x_3^0 \oplus 0 \cdot x_1^1 x_2^0 x_3^0 \oplus 0 \cdot x_1^0 x_2^1 x_3^0 \oplus 1 \cdot x_1^1 x_2^1 x_3^0 \\
 & \oplus 0 \cdot x_1^0 x_2^0 x_3^1 \oplus 0 \cdot x_1^1 x_2^0 x_3^1 \oplus 1 \cdot x_1^0 x_2^1 x_3^1 \oplus 0 \cdot x_1^1 x_2^1 x_3^1.
 \end{aligned}$$

Proceeding in a similar manner for a_1 yields:

a_1						ANF
0	f_0	0		0	f_0	0
1		1	f_0	1	f_1	1
0		0	f_1	0	f_0	0
1		1		0	f_1	0
1	f_1	1		1	f_0	1
0		1	f_0	1	f_1	0
0		0	f_1	1	f_0	1
1		0		1	f_1	0

$$a_1(\mathbf{x}) = 0 \cdot x_1^0 x_2^0 x_3^0 \oplus 1 \cdot x_1^1 x_2^0 x_3^0 \oplus 0 \cdot x_1^0 x_2^1 x_3^0 \oplus 0 \cdot x_1^1 x_2^1 x_3^0 \\ \oplus 1 \cdot x_1^0 x_2^0 x_3^1 \oplus 0 \cdot x_1^1 x_2^0 x_3^1 \oplus 1 \cdot x_1^0 x_2^1 x_3^1 \oplus 0 \cdot x_1^1 x_2^1 x_3^1.$$

Finally, assembling both ANF components we recover the ANF for our generalized Boolean function,

$$f(\mathbf{x}) = 0 \cdot x_1^0 x_2^0 x_3^0 \oplus 0 \cdot x_1^1 x_2^0 x_3^0 \oplus 0 \cdot x_1^0 x_2^1 x_3^0 \oplus 1 \cdot x_1^1 x_2^1 x_3^0 \oplus 0 \cdot x_1^0 x_2^0 x_3^1 \\ \oplus 0 \cdot x_1^1 x_2^0 x_3^1 \oplus 1 \cdot x_1^0 x_2^1 x_3^1 \oplus 0 \cdot x_1^1 x_2^1 x_3^1 + 2(0 \cdot x_1^0 x_2^0 x_3^0 \oplus 1 \cdot x_1^1 x_2^0 x_3^0 \oplus 0 \cdot x_1^0 x_2^1 x_3^0 \\ \oplus 0 \cdot x_1^1 x_2^1 x_3^0 \oplus 1 \cdot x_1^0 x_2^0 x_3^1 \oplus 0 \cdot x_1^1 x_2^0 x_3^1 \oplus 1 \cdot x_1^0 x_2^1 x_3^1 \oplus 0 \cdot x_1^1 x_2^1 x_3^1).$$

The complexity of computing the truth table from the ANF of a Boolean function $f \in \mathcal{B}_n$, is $\mathcal{O}(n2^n)$. The complexity of the butterfly algorithm is also $\mathcal{O}(n2^n)$ [7]. Therefore, the complexity of computing the ANF from the truth table of a generalized Boolean function $f \in \mathcal{GB}_n^q$ (or vice versa), as described above, is $\mathcal{O}(\lceil \log_2 q \rceil n2^n)$.

In a similar manner as was done for Boolean functions in [11], we define the algebraic degree and homogeneity of generalized Boolean function as follows:

Definition 2.10. Given a generalized Boolean function $f \in \mathcal{GB}_n^q$, we define the *algebraic degree* $d^\circ f$ to be the number of variables in the highest order monomial with nonzero coefficients in the ANF of f .

Note that defining the degree of general Boolean functions in this manner is possible due

to the existence and uniqueness of the ANF, which we previously demonstrated.

Definition 2.11. A generalized Boolean function $f \in \mathcal{GB}_n^q$ is said to be *homogeneous* if all of the terms in its ANF are of the same degree.

Seen from the ANF perspective, the simplest Boolean functions are those that are linear or affine (linear function plus a constant). These functions have $d^\circ f = 1$ and are of the form:

$$f(\mathbf{x}) = w_1x_1 \oplus w_2x_2 \oplus \cdots \oplus w_nx_n \oplus w_0.$$

Letting $\mathbf{w} = (w_1, \dots, w_n), \mathbf{x} = (x_1, \dots, x_n) \in \mathbb{V}_n, w_0 \in \mathbb{F}_2$ and denoting $\mathbf{w} \cdot \mathbf{x}$, the usual inner product, we can write: $\mathbf{w} \cdot \mathbf{x} = w_1x_1 \oplus w_2x_2 \oplus \cdots \oplus w_nx_n$ and $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \oplus w_0$. If $w_0 = 0$ then f is linear, otherwise f is affine.

Definition 2.12. We denote the sets of all n -variable *linear* and *affine* functions as \mathcal{L}_n and \mathcal{A}_n , respectively.

Affine functions are important both in coding theory and cryptography. In coding theory affine functions play a key role in Reed-Muller codes of order 1, whereas in cryptography we strive to avoid using affine functions and select instead nonlinear functions whose (cryptographic) behavior is as far as possible from those contained in \mathcal{A}_n [7].

2.3 Fourier Transforms and Generalized Boolean Functions

Definition 2.13. [44] We let $\zeta = e^{2\pi i/q}$ be the complex q -primitive root of unity. To each generalized Boolean function $f(\mathbf{x})$ we associate its *character form*, sometimes also referred to as the sign function in characteristic 2, which is defined as:

$$\hat{f}(\mathbf{x}) = \zeta^{f(\mathbf{x})}.$$

Notice that for $q = 2$, this reduces to the familiar Boolean function character form:

$$\hat{f}(\mathbf{x}) = (-1)^{f(\mathbf{x})}.$$

Definition 2.14. As is customary, given a Boolean function $f(\mathbf{x})$, the *derivative* of $f(\mathbf{x})$ with respect to a vector \mathbf{a} , denoted by $D_{\mathbf{a}}f(\mathbf{x})$, is the Boolean function defined by:

$$D_{\mathbf{a}}f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{a}) \oplus f(\mathbf{x}), \text{ for all } \mathbf{x} \in \mathbb{V}_n.$$

Observe that if $f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{a})$, then $D_{\mathbf{a}}f(\mathbf{x}) = 0$ whereas if $f(\mathbf{x}) \neq f(\mathbf{x} \oplus \mathbf{a})$, then $D_{\mathbf{a}}f(\mathbf{x}) = 1$. Inasmuch, $\sum_{\mathbf{x} \in \mathbb{V}_n} D_{\mathbf{a}}f(\mathbf{x})$ counts the number of input vectors which result in changes to the output values when a change of direction of \mathbf{a} is applied, and can therefore be viewed as a directional derivative.

Definition 2.15. Given a generalized Boolean function $f(\mathbf{x})$, we define the *derivative* $D_{\mathbf{a}}f$ of f with respect to a vector \mathbf{a} to be the generalized Boolean function $D_{\mathbf{a}}f(\mathbf{x})$ by:

$$D_{\mathbf{a}}f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{a}) - f(\mathbf{x}) \text{ for all } \mathbf{x} \in \mathbb{V}_n.$$

Definition 2.16. Given a vector $\mathbf{a} \in \mathbb{V}_n$, we say \mathbf{a} is a *linear structure* of a generalized Boolean function $f(\mathbf{x}) \in \mathcal{GB}_n^q$, if the derivative of $f(\mathbf{x})$ with respect to \mathbf{a} remains constant, that is, if $D_{\mathbf{a}}f(\mathbf{x}) = c \in \mathbb{Z}_q$, for all $\mathbf{x} \in \mathbb{V}_n$.

Definition 2.17. [44] The (normalized) *generalized Walsh–Hadamard transform* of $f \in \mathcal{GB}_n^q$ at any point $\mathbf{u} \in \mathbb{V}_n$ is the complex valued function

$$\mathcal{H}_f(\mathbf{u}) = 2^{-\frac{n}{2}} \sum_{\mathbf{x} \in \mathbb{V}_n} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{u} \cdot \mathbf{x}}.$$

If $q = 2$, we obtain the (normalized) *Walsh–Hadamard transform* of $f \in \mathcal{B}_n$, which will be denoted by W_f [44].

Definition 2.18. [44] The sum

$$\mathcal{C}_{f,g}(\mathbf{z}) = \sum_{\mathbf{x} \in \mathbb{V}_n} \zeta^{f(\mathbf{x}) - g(\mathbf{x} \oplus \mathbf{z})}$$

is the *crosscorrelation* of f and g at \mathbf{z} . The *autocorrelation* of $f \in \mathcal{GB}_n^q$ at $\mathbf{u} \in \mathbb{V}_n$ is $\mathcal{C}_{f,f}(\mathbf{u})$ above, which we denote by $\mathcal{C}_f(\mathbf{u})$.

2.4 Balance and Symmetry

A Boolean function $f \in \mathcal{GB}_n^q$ is balanced if its output values are uniformly distributed. In order for a generalized Boolean function to be balanced, we must have $q = 2^\ell$ for $\ell \leq n$, since the function's q possible output values must be evenly distributed among its 2^n outputs.

Recall that a Boolean function $f \in \mathcal{B}_n$ is balanced if and only if the Hamming weight of its truth table is exactly 2^{n-1} [11].

Lemma 2.19. *If a generalized Boolean function $f(\mathbf{x}) \in \mathcal{GB}_n^{2^\ell}$ is balanced, then its Hamming weight equals $\sum_{i=1}^{2^\ell-1} 2^{n-\ell} = 2^n - 2^{n-\ell}$. Notice that if $\ell = 1$, this reduces to the Boolean function case where the weight of f equals 2^{n-1} .*

Considering Walsh-Hadamard transforms for a moment, we recall from [11] that a Boolean function f is balanced if and only if the Walsh-Hadamard transform,

$$W_f(\mathbf{0}) = 0.$$

In the generalized Boolean function case, we can say the following:

Lemma 2.20. *If a generalized Boolean function f is balanced, then the generalized Walsh-Hadamard transform of f is,*

$$H_f(\mathbf{0}) = \sum_{j=0}^{2^\ell-1} 2^{n-\ell} \zeta^j = 2^{n-\ell} \frac{\zeta^{2^\ell} - 1}{\zeta - 1} = 0.$$

The reader will notice that unlike in the classical Boolean functions case, the preceding criteria for generalized Boolean functions are not biconditional. That is, if a generalized Boolean function is balanced, then the criteria hold. However, for $\ell > 1$, the fact that a generalized function satisfies the Hamming weight or Walsh-Hadamard transform conditions outlined above are necessary, but not sufficient conditions for the function to be balanced. In fact, there are many generalized Boolean functions that satisfy these criteria, yet fail to be balanced.

Theorem 2.21. *A generalized Boolean function $f \in \mathcal{GB}_n^{2^\ell}$ such that $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$ for*

$\mathbf{x} \in \mathbb{V}_n$ is balanced if and only if all of its Boolean functions a_j are balanced, and for each j and h such that $0 \leq j, h \leq k-1$ and $j \neq h$, $d(\mathbf{a}_j, \mathbf{a}_h) = 2^{n-1}$.

Proof. (\Rightarrow) Let $f \in \mathcal{GB}_n^{2^\ell}$ be a balanced generalized Boolean function. Consider the set of 2^ℓ binary vectors $(c_j)_2$ which correspond to the unique output values $c_j \in f(\mathbb{V}_n)$, $0 \leq j \leq 2^\ell - 1$. This set equals \mathbb{V}_{2^ℓ} , which is balanced with respect to the number of 0's and 1's it contains. Moreover, for each column \mathbf{v}_j and \mathbf{v}_h in \mathbb{V}_{2^ℓ} , $d(\mathbf{v}_j, \mathbf{v}_h) = 2^{\ell-1}$. Since each output value of f occurs with frequency $2^{n-\ell}$, this means that each function a_j contains $n - \ell$ copies of \mathbb{V}_{2^ℓ} , thus there are $2^{n-\ell} \cdot 2^{\ell-1} = 2^{n-1}$ 0's and 2^{n-1} 1's and for all Boolean functions, a_j and a_h , where $j \neq h$, $d(\mathbf{a}_j, \mathbf{a}_h) = 2^{n-1}$.

(\Leftarrow) Let $B = \{a_0, a_1, \dots, a_{k-1}\}$ be a collection of k balanced Boolean functions in n variables, such that for all j and h such that $0 \leq j, h \leq k-1$ and $j \neq h$, $d(\mathbf{a}_j, \mathbf{a}_h) = 2^{n-1}$. Let f be a generalized Boolean function $f \in \mathcal{GB}_n^{2^\ell}$ constructed using B such that $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, where $\mathbf{x} \in \mathbb{V}_n$. Consider the composite truth table $A = [\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_{k-1}]$. A consists of 2^n binary row vectors of length k . Each Boolean function is balanced and for any two distinct column vectors (Boolean functions) in A , the pairwise distance between them is 2^{n-1} . Thus, it must be the case that all vectors in \mathbb{V}_{2^ℓ} appear in A with frequency $2^{n-\ell}$. Considering the fact that $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$ and each value $c_j \in f(\mathbb{V}_n)$ is also a binary row vector in A , the result has been demonstrated. \blacksquare

We can obtain a count for the number of balanced generalized Boolean functions by again considering the composite truth table A of the set of Boolean functions $f \in \mathcal{GB}_n^{2^\ell}$. Let $b\mathcal{GB}_n^{2^\ell}$ represent the set of all balanced generalized Boolean functions. There are $\binom{2^n}{2^{n-1}}$ ways in which to select the 2^{n-1} 1's in a_0 . For these 1's, half of the corresponding values in the second truth table, a_1 , must be 1's and the other half must be 0's. There are $\binom{2^{n-1}}{2^{n-2}}$ possible ways to select these 2^{n-2} 1's. Additionally, for the values of a_1 corresponding the remaining 0's in a_0 , half must be 1's and half must be 0's. One can certainly proceed in a similar fashion to get the count, or alternatively, observe that to get a balanced function, one can choose $2^{n-\ell}$ input vectors out of 2^n to assign (via f) the value 0; next choose $2^{n-\ell}$ input vectors out of $2^n - 2^{n-\ell}$ to assign the value 1, etc. That is,

$$|b\mathcal{GB}_n^{2^\ell}| = \binom{2^n}{2^{n-\ell}} \binom{2^n - 2^{n-\ell}}{2^{n-\ell}} \cdots \binom{2^{n-\ell}}{2^{n-\ell}} = \binom{2^n}{2^{n-\ell}, 2^{n-\ell}, \dots, 2^{n-\ell}},$$

the multinomial coefficient with equal parts, each of size $2^{n-\ell}$.

Definition 2.22. A generalized Boolean function $f \in \mathcal{GB}_n^q$ is called *symmetric* if it remains invariant under the full symmetric group S_n .

The task of constructing symmetric generalized Boolean functions $f \in \mathcal{GB}_n^q$, involves partitioning \mathbb{V}_n into q subsets, each of which contains all input vectors of a specific Hamming weight. These q subsets are subsequently mapped to unique values from \mathbb{Z}_q . The number of vectors in \mathbb{V}_n with a given Hamming weight h is $\binom{n}{h}$, thus the cardinality of the subsets within the partition corresponds to the set of binomial coefficients.

In order to establish exactly how many such functions exist, we proceed as follows: First, let $|\mathcal{GB}_n^q|$ represent the total number of symmetric generalized Boolean functions for given n and q . Stirling numbers of the second kind, denoted $\left\{ \begin{smallmatrix} n+1 \\ q \end{smallmatrix} \right\}$, count the number of ways we can partition the set of $n+1$ possible weights of binary input vectors of length n into q nonempty sets. These q nonempty sets must subsequently be mapped to the q possible output values, which can be arranged in $q!$ possible ways. Therefore,

$$|\mathcal{GB}_n^q| = \left\{ \begin{smallmatrix} n+1 \\ q \end{smallmatrix} \right\} q! = \frac{1}{q!} \sum_{i=0}^q (-1)^i \binom{q}{i} (q-i)^{n+1} q! = \sum_{i=0}^q (-1)^i \binom{q}{i} (q-i)^{n+1}.$$

Theorem 2.23. A generalized Boolean function $f \in \mathcal{GB}_n^{2^k}$ such that $f(\mathbf{x}) = a_0(\mathbf{x}) + 2a_1(\mathbf{x}) + \dots + 2^{k-1}a_{k-1}(\mathbf{x})$, is symmetric if and only if each of the Boolean functions $a_i(\mathbf{x}), i \in \{0, 1, \dots, k-1\}$, is symmetric.

Proof. Let $f \in \mathcal{GB}_n^{2^k}$, be a generalized Boolean function such that

$$f(\mathbf{x}) = a_0(\mathbf{x}) + 2a_1(\mathbf{x}) + \dots + 2^{k-1}a_{k-1}(\mathbf{x}),$$

$a_i \in \mathcal{B}_n$. We prove the claim using a counting argument. If a generalized Boolean function \mathcal{GB}_n^q is symmetric, its output remains constant for specific weights of the input \mathbf{x} . There are a total of $n+1$ possible weights for \mathbf{x} . To each of these weights, we have q possible output values. Thus there are q^{n+1} symmetric functions in \mathcal{GB}_n^q . If $q = 2^k$, there are $2^{k(n+1)}$ symmetric functions. Since $f(\mathbf{x}) = a_0(\mathbf{x}) + 2a_1(\mathbf{x}) + \dots + 2^{k-1}a_{k-1}(\mathbf{x})$, we also see

that there are 2^{n+1} possible symmetric Boolean functions for each a_i and a total of $2^{(n+1)k}$ possibilities. These two counts agree and our claim is thus proved. ■

The question of when a Boolean function $f \in \mathcal{B}_n$ is both symmetric and balanced is interesting. Such functions can only exist in the cases where one is able to partition (bisect) the binomial coefficients into two subsets each of sum 2^{n-1} . Although not the main topic of this dissertation, we provide a few remarks regarding the subject here due to the dissertation author's involvement in this research [27].

Letting $\sum_{i=0}^n \delta_i \binom{n}{i} = 0$, where $\delta_i \in \{-1, 1\}$, we can represent $[\delta_0, \dots, \delta_n]$ as a solution to the bisection problem. By the binomial theorem, $\sum_i (-1)^i \binom{n}{i} = (1-1)^n = 0$, hence $\pm[1, -1, 1, -1, \dots]$ is always a solution. Moreover, observe that if n is odd then $[\delta_0, \dots, \delta_{(n-1)/2}, -\delta_{(n-1)/2}, \dots, -\delta_0]$ with $\delta_i \in \{-1, 1\}$ arbitrary chosen, produces $2^{(n+1)/2}$ solutions. These are referred to as trivial solution. Additional, nontrivial bisections occur sporadically. Letting J_n represent the set of bisection solutions for a given n , we have the following theorem:

Theorem 2.24. [27] *If p is a prime number, then $J_{p-1} = 2$.*

Proof. The statement is obviously true if $p = 2$, so we may assume that p is an odd prime. We let $n = p - 1$ and observe that $n \equiv -1 \pmod{p}$. We want to show that $\binom{n}{j} \equiv (-1)^j \pmod{p}$, for every $j \in \{0, 1, \dots, n\}$. This is clearly true for $j = 0$. Since, every $j \in \{1, \dots, n\}$ has an inverse modulo p , we have for $j \in \{1, \dots, n\}$

$$\begin{aligned} \binom{n}{j} &\equiv \frac{n(n-1) \cdots (n-j+1)}{j!} \\ &\equiv \frac{(-1)(-2) \cdots (-1-j+1)}{j!} \equiv (-1)^j \pmod{p}. \end{aligned}$$

Hence, if $[\delta_0, \dots, \delta_n]$ a solution of the bisection problem is

$$0 = \sum_{j=0}^n \delta_j \binom{n}{j} \equiv \sum_{j=0}^n (-1)^j \delta_j \pmod{p},$$

but the number

$$\Delta := \sum_{j=0}^n (-1)^j \delta_j \equiv 0 \pmod{p}$$

is an odd number ($n+1 = p$ is an odd prime) satisfying

$$|\Delta| \leq \sum_{j=0}^n |(-1)^j \delta_j| = \sum_{j=0}^n 1 = n+1 = p. \quad (2.2)$$

Because Δ cannot be zero, the only possible values of Δ are p or $-p$. Then the equality $|\Delta| = p = n+1$ in (2.2), forces $\delta_j = \pm(-1)^j$, for all j . Therefore, we have only the two trivial solutions, that is, $J_n = 2$ [27]. ■

Using the Hamming high performance computer (HPC) at the Naval Postgraduate School, and a parallel computer program written in Julia, (see appendix A2), we were able to exhaustively search for nontrivial binomial bisections for $n \leq 51$ [27]. We verified the computational data previously provided by [10] and [21] for $n < 37$, and obtained additional results for $37 \leq n \leq 51$. These results have been included in Appendix A.1 and the number of nontrivial solutions appear as A200147 in the Online Encyclopedia of Integer Sequences.

Looking at the bisection solution data in appendix A.1 we observed some additional patterns. Using some identities, which were first pointed out by Jefferies [21], along with solutions to diophantine equations, we were able to produce some infinite classes of integers admitting nontrivial bisections. We present the following from our research without proof. Additional discourse on this topic along with the proof of the following theorem can be found in the paper entitled *Bisecting binomial coefficients* which recently appeared in the journal *Discrete Applied Mathematics* [27].

Theorem 2.25. [27] *The following hold:*

1. If $n = k^2 - 2$, $k \geq 4$ even, then $J_n \geq 10$, $J_{n-1} \geq 2^{\frac{n+1}{2}} + 2^{\frac{n+1}{2}-3}$ (tight).
2. If $k \equiv 0, 1 \pmod{3}$ and $n = \frac{F_{4k+1} + 2F_{4k} - 6}{5}$, then $J_n \geq 2^{\frac{n+1}{2}} + 2^{\frac{n-3}{2}}$.
3. Let $n = 4k^2 + 16k + 13$, $k \geq 0$. Then, there are at least $2^{(n+1)/2-3}$ nontrivial bisections for the binomial coefficients $\left\{ \binom{n}{j} \right\}_{0 \leq j \leq n}$, and so, $J_n \geq 2^{\frac{n+1}{2}} + 2^{\frac{n-1}{2}}$.

Based on related search data, we make the following conjecture regarding the impossibility of further sub-dividing the binomial coefficients into equal parts.

Conjecture 2.26. *There are no 2^k -sections of the binomial coefficients for $k > 1$.*

Should a proof of this conjecture emerge, it would mean that symmetric and balanced generalized Boolean functions do not exist.

CHAPTER 3:

Correlation Immune Generalized Boolean Functions

The Devil is in the details, but so is salvation.

Hyman G. Rickover ★★ ★

3.1 Introduction

Siegenthaler first described the correlation attack in 1984 [40]. This type of known plaintext attack provides cryptanalysts with a method of attacking stream ciphers which are generated using multiple Linear feedback shift registers (LFSRs) and a nonlinear combiner which is plagued by a poorly chosen Boolean function. Correlation attacks involve careful examination of input vectors and their associated functional outputs in order to determine whether the value of a single bit, or the values of a subsets of bits in the input vector exert greater influence over the output than others. If this is the case, attackers can use this information to surmise something about the structure of the underlying Boolean function as well as the outputs of the LFSRs. Cusick and Stănică provide an example of such a poorly chosen function in [11, p. 58] that we, for illustrative purposes, provide here.

Example 3.1. Consider the following 3-variable Boolean function $f(\mathbf{x}) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3$ and its associated truth table:

Input	000	001	010	011	100	101	110	111
Output	0	0	0	1	0	1	1	1

To determine whether or not the value of a single input bit exerts an undue influence over the output, we use the truth table and compute conditional probabilities for each bit of the input vectors, \mathbf{x} . For example, the probability that the first bit x_1 is 0 given the fact that the function's output equals 0 is

$$Pr(x_1 = 0 | f(\mathbf{x}) = 0) = \frac{Pr(x_1 = 0 \cap f(\mathbf{x}) = 0)}{Pr(f(\mathbf{x}) = 0)} = \frac{3/8}{4/8} = 3/4.$$

Proceeding similarly, we calculate the conditional probabilities for each of the remaining possibilities and obtain the results listed in table 3.1.

Table 3.1: Conditional probability table for a Boolean function

Conditional Prob. Given $f(\mathbf{x}) = 0$	Conditional Prob. Given $f(\mathbf{x}) = 1$
$Pr(x_1 = 0 f(\mathbf{x}) = 0) = 3/4$ $Pr(x_1 = 1 f(\mathbf{x}) = 0) = 1/4$	$Pr(x_1 = 0 f(\mathbf{x}) = 1) = 1/4$ $Pr(x_1 = 1 f(\mathbf{x}) = 1) = 3/4$
$Pr(x_2 = 0 f(\mathbf{x}) = 0) = 3/4$ $Pr(x_2 = 1 f(\mathbf{x}) = 0) = 1/4$	$Pr(x_2 = 0 f(\mathbf{x}) = 1) = 1/4$ $Pr(x_2 = 1 f(\mathbf{x}) = 1) = 3/4$
$Pr(x_3 = 0 f(\mathbf{x}) = 0) = 3/4$ $Pr(x_3 = 1 f(\mathbf{x}) = 0) = 1/4$	$Pr(x_3 = 0 f(\mathbf{x}) = 1) = 1/4$ $Pr(x_3 = 1 f(\mathbf{x}) = 1) = 3/4$

Examining the table we see that if the function's output is zero, the probabilities that each respective input bit, x_1, x_2 , and x_3 , equal zero are all .75. From a cryptographic perspective this is highly undesirable! Armed with this information and known plaintext, an adversary readily obtains information about the outputs of the LFSRs, which in turn can be used to launch an attack on each LFSR, thereupon recovering the keystream of the system.

To avoid this unfortunate situation, we need to be more circumspect in how we go about choosing our Boolean function. To be in a position to select more wisely we initially adopt a "black box" view of the problem and consider input vectors and the output values to which they are mapped. We partition the set of input vectors \mathbb{V}_n into two sets V_0 and V_1 , such that $\forall \mathbf{x} \in V_0, f(\mathbf{x}) = 0$ and $\forall \mathbf{x} \in V_1, f(\mathbf{x}) = 1$. Clearly, in order to not give away any information to a would-be-attacker, for $i = 1, 2, 3$, the conditional probabilities for all $\mathbf{x} \in V_0$, $Pr(x_i = 0|f(\mathbf{x}) = 0) = Pr(x_i = 1|f(\mathbf{x}) = 0) = 1/2$. Consequently, we recognize that $|V_0| > 1$. If this were not the case, the output value 0 would appear only once in the function's truth table and it would be associated with a single input vector \mathbf{x} . This in turn would result in the probability $Pr(x_i = 0|f(\mathbf{x}) = 0)$, for each respective index, $i, 1 \leq i \leq 3$, being equal to either 1 or 0. It is, however, possible for $|V_0|$ to equal 2 and ensure that the necessary conditional probabilities hold. Partitioning \mathbb{V}_n using complementary input vectors we can construct a Boolean function $f : \mathbb{V}_n \rightarrow \mathbb{F}_2$ with the desired conditional probability properties we seek. Partition \mathbb{V}_n into two subsets, S_0, S_1 , such that $\forall \mathbf{x} \in S_j, \bar{\mathbf{x}} \in S_j$ and $f(\mathbf{x}) = f(\bar{\mathbf{x}}) = j$,

where $j = 0, 1$. To see that this will produce the desired result, consider the following: For each pair of vectors, $\mathbf{x}, \bar{\mathbf{x}} \in S_j$ and each bit x_i , where $i = 1, 2, \dots, n$, there is one vector where $x_i = 0$ and one vector where $x_i = 1$. This means that if S_j contains m pairs of complementary vectors, then for each i , there are m vectors where $x_i = 0$ and m vectors where $x_i = 1$, which in turn yields

$$Pr(x_i = 0 | f(\mathbf{x}) = 0) = Pr(x_i = 1 | f(\mathbf{x}) = 0) = \frac{m/|\mathbb{V}_n|}{|S_j|/|\mathbb{V}_n|} = \frac{m/2^n}{2m/2^n} = \frac{1}{2}.$$

Equipped with this new-found insight, we tailor the following truth table for our new Boolean function:

Input	000	001	010	011	100	101	110	111
Output	1	1	1	0	0	1	1	1

Converting the truth table into ANF yields the Boolean function $f(\mathbf{x}) = 1 \oplus x_2x_3 \oplus x_1 \oplus x_1x_3 \oplus x_1x_2$. We subsequently compute the conditional probabilities given in Table 3.2, and verify that our analysis did in fact render a Boolean function with the desired properties.

Table 3.2: Conditional probability table for an order 1 correlation immune Boolean function

Conditional Prob. Given $f(\mathbf{x}) = 0$	Conditional Prob. Given $f(\mathbf{x}) = 1$
$Pr(x_1 = 0 f(\mathbf{x}) = 0) = 1/2$	$Pr(x_1 = 0 f(\mathbf{x}) = 1) = 1/2$
$Pr(x_1 = 1 f(\mathbf{x}) = 0) = 1/2$	$Pr(x_1 = 1 f(\mathbf{x}) = 1) = 1/2$
$Pr(x_2 = 0 f(\mathbf{x}) = 0) = 1/2$	$Pr(x_2 = 0 f(\mathbf{x}) = 1) = 1/2$
$Pr(x_2 = 1 f(\mathbf{x}) = 0) = 1/2$	$Pr(x_2 = 1 f(\mathbf{x}) = 1) = 1/2$
$Pr(x_3 = 0 f(\mathbf{x}) = 0) = 1/2$	$Pr(x_3 = 0 f(\mathbf{x}) = 1) = 1/2$
$Pr(x_3 = 1 f(\mathbf{x}) = 0) = 1/2$	$Pr(x_3 = 1 f(\mathbf{x}) = 1) = 1/2$

The function which we constructed above is referred to as a correlation immune (order 1) function. Order 1 refers to the fact that it only satisfies the conditional probability requirements for a single bit. It is of course possible to consider larger subsets of bits in the input vectors of a function. In the above case, f fails in multiple instances when we consider values assignments of the $\binom{3}{2}$ two bit subsets. For example,

$$Pr(x_1 = 0, x_3 = 0 | f(\mathbf{x}) = 1) = \frac{2/8}{6/8} = 1/3.$$

Correlation attacks take advantage of differences in the conditional probabilities between subsets of input vector bits and the associated outputs of a function. Seen from this "black box" perspective, it is of little consequence whether the function's output is binary or a subset of values from some other ring \mathbb{Z}_q ($q > 2$). If a cryptographer hopes to render a function immune to this adversarial technique, he must ensure that balanced conditional probabilities exist for all values of the image of f . Thus far, we have considered output values $c \in \mathbb{F}_2$, but we could have just as well considered the output values $c \in \mathbb{Z}_q$. As such, there is a very natural extension of the concept of correlation immunity into the domain of generalized Boolean functions. With this in mind, we extend Cusick and Stănică's definition of correlation immunity from [11, p. 55].

Definition 3.2. A generalized Boolean function $f \in \mathcal{GB}_n^q$ is said to be *correlation immune of order t* , with notation $CI(t)$, $1 \leq t \leq n$, if for any fixed subset of t variables the probability that, given the value of $f(\mathbf{x})$, the t variables have any fixed set of values, is always 2^{-t} , no matter what the choice of the fixed set of t values is.

When exploring the notion of correlation immunity for generalized Boolean functions, a fitting place to begin is perhaps by contemplating just how many output values, $c \in \mathbb{Z}_q$, a correlation immune generalized Boolean function could possibly achieve.

Theorem 3.3. *If $f \in \mathcal{GB}_n^q$ is a CI (order 1) generalized Boolean function, then the number of occurrences of each output value $c \in \mathbb{Z}_q$ that f achieves is even.*

Proof. Let $f \in \mathcal{GB}_n^q$ be a CI (order 1) generalized Boolean function. Let $\mathbf{x} = (x_n, \dots, x_1) \in \mathbb{V}_n$. Suppose S instances of a specific output value, $c \in \mathbb{Z}_q$, occur in the truth table of f . Let $V_c \subset \mathbb{V}_n$, represent the set of all vectors \mathbf{x} such that $f(\mathbf{x}) = c$. For each $i = 1, 2, \dots, n$, let $V_{(0,i)} \subset V_c$ be the subset of vectors such $x_i = 0$ and $f(\mathbf{x}) = c$ and let $V_{(1,i)} \subset V_c$ be the subset of vectors such that $x_i = 1$ and $f(\mathbf{x}) = c$. Then, since f is CI(1), for each $i = 1, 2, \dots, n$ we have

$$Pr(x_i = 0 | f(\mathbf{x}) = c) = \frac{Pr(x_i = 0 \cap f(\mathbf{x}) = c)}{Pr(f(\mathbf{x}) = c)} = \frac{\frac{|V_{(0,i)}|}{2^n}}{\frac{|V_c|}{2^n}} = \frac{|V_{(0,i)}|}{S} = \frac{1}{2} \implies |V_{(0,i)}| = \frac{S}{2}$$

and

$$Pr(x_i = 1 | f(\mathbf{x}) = c) = \frac{Pr(x_i = 1 \cap f(\mathbf{x}) = c)}{Pr(f(\mathbf{x}) = c)} = \frac{\frac{|V_{(1,i)}|}{2^n}}{\frac{|V_c|}{2^n}} = \frac{|V_{(1,i)}|}{S} = \frac{1}{2} \implies |V_{(1,i)}| = \frac{S}{2}.$$

For each i , x_i is either 0 or 1, so $V_{(0,i)}$ and $V_{(1,i)}$ are mutually exclusive. Moreover, $|V_{(0,i)}| = |V_{(1,i)}|$ for all i , therefore $2 \cdot |V_{(0,i)}| = 2 \cdot |V_{(1,i)}| = S$, and the result is thus proven. ■

Corollary 3.4. *Let $f \in \mathcal{G}\mathcal{B}_n^q$ be a correlation immune (order 1) generalized Boolean function and let $f(\mathbb{V}_n)$ be the image of f . Then $|f(\mathbb{V}_n)| \leq 2^{n-1}$.*

Proof. The result is an immediate consequence of Theorem 3.3. Let $f \in \mathcal{G}\mathcal{B}_n^q$ be a $CI(1)$ generalized Boolean function. Since the number of occurrences of each distinct output value $c \in f(\mathbb{V}_n)$ must be divisible by 2, the maximum number of output values is therefore $\frac{|\mathbb{V}_n|}{2} = \frac{2^n}{2} = 2^{n-1}$. ■

Remark 3.5. We have already demonstrated how one can create a $CI(1)$ generalized Boolean function $f \in \mathcal{G}\mathcal{B}_n^q$, by ensuring that $f(\mathbf{x}) = f(\bar{\mathbf{x}})$, for all $\mathbf{x} \in \mathbb{V}_n$. By assigning a distinct value, $c \in \mathbb{Z}_q$, for each vector pair $\mathbf{x}, \bar{\mathbf{x}}$ we achieve the above stated upper bound.

3.2 Correlation Immune Constructions

There are numerous ways in which to construct correlation immune (order 1) Boolean functions. In addition to the so-called “folklore” construction, that we have touched upon, a method which we refer to as the “complementation construction” works well. In this case we create correlation immune (order 1) Boolean functions $f \in \mathcal{B}_n$ using the following algorithm:

Algorithm 2 $CI(1)$ Complementation Construction for Boolean Functions

- 1: Write the truth table of f , in which the binary vectors of length n are in lexicographic order.
 - 2: Label the first 2^{n-1} entries of the truth table with 2^{n-2} 0's and 2^{n-2} 1's in any order desired.
 - 3: Label the remaining 2^{n-1} entries of the truth table by copying the first 2^{n-1} entries of the truth table into the the second half of the truth table and then complement each of these entries.
-

Example 3.6. Consider the following truth table of a correlation immune order 1 function $f \in \mathcal{B}_4$, which was created using the complementation algorithm. In order to highlight the complementation process and motivate the subsequent proof of correctness of the algorithm, we include the set of input vectors, \mathbb{V}_4 , and place the two halves of the truth table side-by-side.

Table 3.3: A $CI(1)$ Boolean function $f \in \mathcal{B}_4$

\mathbb{V}_4	f	\mathbb{V}_4	f
0000	0	1000	1
0001	0	1001	1
0010	1	1010	0
0011	0	1011	1
0100	1	1100	0
0101	1	1101	0
0110	1	1110	0
0111	0	1111	1

Proof of Correctness of the $CI(1)$ Boolean Function Complementation Construction:

Suppose we create a Boolean function $f \in \mathcal{B}_n$ using the preceding algorithm. To show that the algorithm indeed renders a correlation immune (order 1) function, we argue as follows: Partition the set of input vectors \mathbb{V}_n into two sets, V_0 and V_1 , such that for all $\mathbf{x} \in V_j$, $f(\mathbf{x}) = j$, where $j = 0$ or $j = 1$. Let $V_{j|_{n-1}}$ represent the set of sub-vectors of the $n - 1$ least significant bits of V_j . Since the second half of the truth table is a complemented copy of the first half, $V_{j|_{n-1}} = \mathbb{V}_{n-1}$ for both $j = 0$ and $j = 1$. Now, for each column i from 1 to $n - 1$, we know that \mathbb{V}_{n-1} is balanced (contains an equal number of 0's and 1's), therefore it must also be the case that each set $V_{j|_{n-1}}$, where $j = 0$ or 1 , is also balanced with respect to the first $n - 1$ columns. Moreover, the algorithm required that the first half of the truth table was balanced, which in turn ensures that the n^{th} column is also balanced in both V_0 and V_1 . Consequently, for all i from 1 to n , $Pr(\mathbf{x}_i = 0 | f(\mathbf{x}) = 0) = 1/2$, thus demonstrating that the function is correlation immune (order 1).

The complementation algorithm allows us to create a great many $CI(1)$ Boolean functions.

Unfortunately, this construction method is not well suited for building correlation immune (order 1) generalized Boolean functions. For this, we require a more general technique which partitions \mathbb{V}_n into appropriate subsets of input vectors, which each can in turn be mapped to different output values of \mathbb{Z}_q . To accomplish this task we generalize the “folklore” construction. This method required that the function be such that for all $\mathbf{x} \in \mathbb{V}_n$, $f(\mathbf{x}) = f(\bar{\mathbf{x}})$. In other words, $f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{a})$, where $wt(\mathbf{a}) = n$. Recall that a vector $\mathbf{a} \in \mathbb{V}_n$ is a linear structure of a function f , if the derivative of f with respect to \mathbf{a} remains constant. In other words, for all CI(1) functions f , which were created using the “folklore” construction, $\mathbf{a} = 111 \dots 1$, is a linear structure of f . There is, *per se*, no reason why we must choose this linear structure. We might just have well chosen another linear structure.

Algorithm 3 *CI(1) generalized Boolean function construction*

- 1: Pick a vector, $\mathbf{a} \in \mathbb{V}_n$, such that $0 \leq \kappa \leq n - 1$ and $wt(\mathbf{a}) = n - \kappa$.
 - 2: For all $\mathbf{x} \in \mathbb{V}_n$, pair \mathbf{x} with $\mathbf{x}' = \mathbf{x} \oplus \mathbf{a}$.
 - 3: Vectors within each of the 2^{n-1} pairs, agree in κ positions. If $\kappa = 0$, map each pair to any desired output value, \mathbb{Z}_q . Otherwise, for each pair of vectors, combine it with a corresponding pair of vectors which differ with respect to the bits found at the indices where 0's occur in \mathbf{a} .
 - 4: Finally, map each of the 2^{n-2} sets of four vectors to any output value, \mathbb{Z}_q .
-

Proof of Correctness of the *CI(1)* Generalized Boolean Function Construction: Suppose we create a Boolean function $f \in \mathcal{GB}_n^q$, where $1 \leq q \leq 2^{n-1}$, using the above described algorithm. The set of input vectors \mathbb{V}_n is a linear vector space, so for every $\mathbf{a} \in \mathbb{V}_n$, using the procedure whereby we for all $\mathbf{x} \in \mathbb{V}_n$, pair \mathbf{x} with $\mathbf{x}' = \mathbf{x} \oplus \mathbf{a}$, uniquely partitions \mathbb{V}_n into 2^{n-1} pairs of vectors. Let κ represent the number of zeros contained in \mathbf{a} , so $wt(\mathbf{a}) = n - \kappa$. Then the vectors, \mathbf{x} and \mathbf{x}' , within each pair agree in κ of the n index positions. If $\kappa = 0$, each vector pairs can be mapped to any output value $c \in \mathbb{Z}_{n-1}$. (This is the “folklore” construction.) If on the other hand $\kappa > 0$, then there are 2^κ possible combinations for the bits in the κ indices which correspond to where zeros occur in \mathbf{a} . However, since we have partitioned \mathbb{V}_n , and each column of \mathbb{V}_n , contains an equal number of 0's and 1's, there must be $2^{n-1-\kappa}$ vector pairs which contain each of the 2^κ possibilities. This in turn guarantees that for every vector pair within the partition, it is always possible to combine two corresponding pairs of vectors which disagree with respect to each of the bits

found at the indices where zeros occur in \mathbf{a} . In fact, for a given \mathbf{a} , there are a total of

$$(2^{n-1-\kappa}!)^{2^{\kappa-1}}$$

such groupings. Once one of these groupings has been carried out, we have ensured that each set of four vectors contain an equal number of 0's and 1's with respect to those indices. For the remaining indices, \mathbf{a} contained all ones, which ensured that each of the 2^{n-1} vector pairs already contained a balance of 0's and 1's in these positions. Thus, by subsequently mapping each set of four vectors to an output value $c \in \mathbb{Z}_{n-2}$, the algorithm guarantees that for all i from 1 to n , $Pr(\mathbf{x}_i = 0 | f(\mathbf{x}) = c) = 1/2$. Hence the function is correlation immune (order 1).

Example 3.7. Suppose we wish to construct a CI(1) generalized Boolean function $f \in \mathcal{GB}_4^q$, where $1 \leq q \leq 4$. Rather than using the all ones vector to partition \mathbb{V}_n , we select instead the vector $\mathbf{a} = 1010$. Letting κ represent the number of zeros in \mathbf{a} , we then have $\kappa = 2$ with zeros occurring at index 1 and 3 (indexing from least to most significant bit). For each $\mathbf{x} \in \mathbb{V}_4$, we pair \mathbf{x} with $\mathbf{x}' = \mathbf{x} \oplus \mathbf{a}$. Doing so yields the following partition:

0000	0010	0100	0110	0001	0011	0101	0111
1010	1000	1110	1100	1001	1001	1111	1101

Since $\kappa = 2$, there are $2^2 = 4$ possible two bit combinations for the bits located at index 1 and 3. Moreover, there are $2^{n-1-\kappa} = 2$ pairs of vectors which contain each of the possible 4-bit combinations at indices 1 and 3. We now combine each pair of vectors with a corresponding pair which disagrees with respect to the bits at index 1 and 3. There are a total of $(2^{n-1-\kappa}!)^{2^{\kappa-1}} = (2!)^2 = 4$ possible ways this can be accomplished. Finally, we map each of the $2^{n-2} = 4$ sets of vectors to 4 possible output values from \mathbb{Z}_4 . Therefore, based on our selection of \mathbf{a} , there are a total of $4^4 = 256$ possible correlation immune (order 1) generalized Boolean functions which can be constructed using this algorithm. We list one such possible function in Table 3.4:

Table 3.4: A $CI(1)$ generalized Boolean function $f \in \mathcal{GB}_4^4$

\mathbb{V}_4	f
0000	0
0001	3
0010	2
0011	1
0100	1
0101	2
0110	3
0111	0
1000	2
1001	1
1010	0
1011	3
1100	3
1101	0
1110	1
1111	2

3.3 A Higher Order Correlation Immune Construction

The above algorithm enables us to construct a large class of order 1 correlation immune generalized Boolean functions. Although higher order correlation immune functions are less prevalent, we would none-the-less like to devise an algorithm with which we can construct correlation immune generalized Boolean functions of higher order. Before proceeding we must first introduce the following:

Definition 3.8. [11, p. 72] An $m \times n$ array with entries from a set of s elements is called an *orthogonal array* of size m with n constraints, s levels, strength t , and index r , if any set of t columns of the array contain all s^t possible row vectors exactly r times. We will throughout this dissertation denote such orthogonal arrays by $OA(m, n, s, t)$.

There is a close connection between correlation immune Boolean functions and orthogonal arrays. Camion, et al. first corresponded on this topic in 1992 [3].

Theorem 3.9. *Every partition P of \mathbb{V}_n which consists of q binary orthogonal arrays, each of index 1 and strength t , can be used to construct a correlation immune (order t) generalized Boolean function $f \in \mathcal{GB}_n^q$, and every order t correlation immune generalized Boolean function $f \in \mathcal{GB}_n^q$ generates a partition P of \mathbb{V}_n , where P consist of q binary orthogonal arrays, each of index 1 and strength t .*

Proof. (\Rightarrow) Let P be a partition of \mathbb{V}_n comprised of q binary orthogonal arrays O_j , $0 \leq j \leq q-1$, each of index 1 and strength t . For all j and all vectors $\mathbf{x} \in O_j$, map $\mathbf{x} \rightarrow c_j$, where each value c_j is a distinct value in \mathbb{Z}_q . This creates a generalized Boolean function $f \in \mathcal{GB}_n^q$. By Definitions 3.8, any set of t columns of each O_j contains all 2^t possible row vectors once. Given the stipulated mapping, this in turn means that according to Definition 3.2, f is an order t correlation immune generalized Boolean function.

(\Leftarrow) Let $f \in \mathcal{GB}_n^q$ be an order t correlation immune generalized Boolean function. For each distinct output value $c_j \in \mathbb{Z}_q$, $0 \leq j \leq q-1$, partition \mathbb{V}_n into q subsets O_j such that $O_j = \{\mathbf{x} \in \mathbb{V}_n : f(\mathbf{x}) = c_j\}$. The function f is correlation immune of order t , therefore according to Definition 3.2, for any fixed subset of t input vector variables, x_i , $1 \leq i \leq n$, the probability that, for $f(\mathbf{x}) = c_j$, the t variables have any fixed set of values is 2^{-t} . Thus according to Def. 3.8, each O_j must be an index 1, strength t binary orthogonal array. ■

Consequently, although not mentioned at the time, the subsets of \mathbb{V}_n which were created in the constructions of Algorithms 2 and 3 were in fact binary orthogonal arrays of index and strength 1. It is interesting to note that \mathbb{V}_n is itself an orthogonal array of strength n . This is the reason why all constant functions are (order n) correlation immune.

Lemma 3.10. *Let O be an $OA(m, n, 2, t)$ binary orthogonal array. Complementing any column, i , $1 \leq i \leq n$, of O produces another $OA(m, n, 2, t)$ binary orthogonal array.*

Proof. Let O be an $OA(m, n, 2, t)$ binary orthogonal array. Suppose by way of contradiction that we complement a column, i , $1 \leq i \leq n$, of O and that the resultant array, O' is no longer an orthogonal array. If O' is not an orthogonal array, it must be the case that there exist some set of t columns for which one of the 2^t possible binary row vectors occurs with a frequency

less than r . Now, O was an orthogonal array, so for all possible combinations of t columns, each of the 2^t possible binary row vectors in O occurred each with frequency r . The only changes made to O , took place in column i . Therefore, if one of the 2^t possible row vectors in O' occurs with a frequency less than r , it must be the case that there exist an unequal number of 0's and 1's in column i of O' . However, since column i in O' is the complement of column i from O , this would mean that an imbalance of 0's and 1's existed in O , which in turn would mean that one of the 2^t possible binary row vectors of O also occurred with a frequency less than r . This contradicts the fact that O is an orthogonal array. We therefore conclude that complementing any column of an orthogonal array, $OA(m, n, 2, t)$, results in another orthogonal array, $OA(m, n, 2, t)$. ■

Example 3.11. Consider the following 4×3 binary array, X , along with all possible combinations of two of its columns:

x_1	x_2	x_3	x_1	x_2	x_1	x_3	x_2	x_3
0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	1	1	1
1	0	1	1	0	1	1	0	1
1	1	0	1	1	1	0	1	0

For every possible combination of 2 columns of X , the row vectors 00, 01, 10, and 11 all occur with frequency 1. Consequently, this is a $OA(4, 3, 2, 2)$ orthogonal array of index 1. Moreover, according to Lemma 3.10, complementing any column of X , for example column number 3, produces yet another $OA(4, 3, 2, 2)$ orthogonal array, X' :

x_1	x_2	\bar{x}_3
0	0	1
0	1	0
1	0	0
1	1	1

There also exists a relationship between orthogonal arrays and error correcting codes [2], [12], [19]. This connection is due to the fact that the codewords of an error correcting code can be used as the rows of an orthogonal array, or conversely the rows of an orthogonal

array can be regarded as codewords of an error correcting code. For purposes which soon shall become clear, our construction will make use of orthogonal arrays created using linear error-correcting codes. Neither error-correcting codes nor orthogonal arrays are the focus of this dissertation. However, due to central role which these topics play in our construction method of high order correlation immune generalized Boolean function, we deem it prudent to include a few basic definitions, lemmas, and theorems for the benefit of readers unfamiliar with these topics. Rather than restating and reproving these results, much of this introductory material has been taken from Chapter 4 of Hedayat, Sloane, and Stufken's excellent monograph on orthogonal arrays [19]. For consistency's sake, we retain our finite field notation \mathbb{F}_s , where s is power of a prime, rather than adopt the authors' notation of $GF(s)$ found in the original publication.

Definition 3.12. [19, p. 65] An *error correcting code* C of length n , size m , minimum pairwise Hamming distance between distinct codewords of d , and which is defined over an alphabet S of size $|S| = s$, is denoted $(n, m, d)_s$. To any such code we associate the $m \times n$ array whose rows are the codewords of C . This array is an orthogonal array $OA(m, n, s, t)$ for some t .

Definition 3.13. [19, p. 63] A code C of length n is said to be *linear* if the codewords are distinct and C is a vector subspace of \mathbb{F}_s^n , thus C has size $m = s^\ell$ for some non negative integer $0 \leq \ell \leq n$. Additionally, the *minimum distance* d for a linear code is equal to the minimal Hamming weight of any nonzero codeword.

Definition 3.14. [19, p. 40] An orthogonal array is *simple* if the rows of the array are distinct.

Definition 3.15. [19, p. 40] Let s be a prime power. An orthogonal array $OA(m, n, s, t)$ with levels from \mathbb{F}_s is said to be *linear* if it is simple and if, when considered as n -tuples from \mathbb{F}_s , its m rows form a vector space over \mathbb{F}_s .

Lemma 3.16. [19, p. 65] *The orthogonal array associated with a code is linear if and only if the code is linear.*

Proof. This follows immediately from the preceding definitions of linearity. ■

A linear (m, n) -code can be concisely described using an $m \times n$ generator matrix, G , in which the rows of the matrix form a basis for the code. The code C , then consists of all vectors $\mathbf{u} = \mathbf{x}G$, where \mathbf{x} runs through all $\mathbf{x} \in S^n$ [19, p. 64].

Example 3.17. The $(7, 8, 4)_2$ code can be represented using the following generator matrix:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Each of the $2^3 = 8$ codewords can then be obtained by using the encoding function, $E(\mathbf{x}) = \mathbf{x}G$, where $\mathbf{x} \in \mathbb{V}_3$. For example, the codeword associated with the vector $\mathbf{x} = 010$ is:

$$E(010) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

For each linear code C , there exists an associated linear code called its *dual*, which we denote by C^\perp . This code consists of all vectors $\mathbf{v} \in S^n$ such that

$$\mathbf{u}\mathbf{v}^T = 0, \quad \forall \mathbf{u} \in C.$$

For example, the dual of the $(7, 8, 4)_2$ code given in Example 3.17 is a $(7, 16, 3)_2$ Hamming code. We refer to a code which is its own dual as a *self-dual* code. The distance of the dual code of C is further denoted d^\perp .

Lemma 3.18. [19, p. 54] *Let A be an orthogonal array $OA(m, n, s, t)$ with entries from \mathbb{F}_s . Then any t columns of A are linearly independent over \mathbb{F}_s .*

Proof. $m \times 1$ vectors v_1, \dots, v_t with components from a ring R are said to be linearly independent over R if the relation

$$c_1 v_1 + \dots + c_t v_t = 0, \quad c_1, \dots, c_t \in R, \quad (3.1)$$

implies that $c_1 = \dots = c_t = 0$. An equivalent condition is that the matrix $[v_1 \dots v_t]$ has rank t over R . Now let v_1, \dots, v_t be any t columns of A , and suppose (3.1) holds. There is a row vector i with the first entry equal to 1 and others 0. Then (3.1) implies $c_1 = 0$. Similarly $c_2 = \dots = c_t = 0$ [19, p. 54]. ■

Lemma 3.19. [19, p. 54] *Let A be an $m \times n$ matrix whose rows form a linear subspace of \mathbb{F}_{s^k} . If any t columns of A are linearly independent over \mathbb{F}_s , then A is an orthogonal array $OA(m, n, s, t)$.*

Proof. Suppose $m = s^\ell$, and let G be an $\ell \times n$ generator matrix for A , so that the rows of A consist of all n -tuples ξG , where $\xi = (\xi_1, \dots, \xi_\ell)$, $\xi_i \in \mathbb{F}_s$. Choose t columns of A , and let G_1 be the corresponding $\ell \times t$ submatrix of G . Clearly the columns of G_1 are linearly independent. The number of times that a t -tuple z appears as a row in these t columns of A is equal to the number of ξ such that

$$\xi G_1 = z.$$

Since G_1 has rank t , this number is $s^{\ell-t}$, for all z . Therefore A is an orthogonal array of strength t [19, p. 54]. ■

We are now in a position to introduce the following important theorem which establishes the connection between orthogonal arrays and linear codes and specifies how the strength of a linear orthogonal array is related to the associated linear code. Although we use Hedayat's proof of the theorem here, the theorem itself is attributed to Bose who included the result in his 1961 paper [2].

Theorem 3.20. [19, p. 66] *If C is a $(n, m, d)_s$ linear code over \mathbb{F}_s with dual distance d^\perp then the codewords of C form rows of an orthogonal array $OA(m, n, s, d^\perp - 1)$ with entries from \mathbb{F}_s . Conversely, the rows of a linear orthogonal array $OA(m, n, s, t)$ over \mathbb{F}_s form a $(n, m, d)_s$ linear code over \mathbb{F}_s with dual distance $d^\perp \geq t + 1$. If the orthogonal array has strength t but not $t + 1$, d^\perp is precisely $t + 1$.*

Proof. (\Rightarrow) Suppose C is a $(n, m, d)_s$ linear code over \mathbb{F}_s with dual distance d^\perp . Let A be the array formed by the codewords of C . Any $d^\perp - 1$ columns of A must be linearly independent over \mathbb{F}_s , or else there would be a codeword of weight less than d^\perp in the dual code, which would contradict the hypothesis that d^\perp is the minimal nonzero distance in the dual code. By Lemma 3.19, A is an $OA(m, n, s, d^\perp - 1)$.

(\Leftarrow) Conversely, let C be the code associated with a linear $OA(m, n, s, t)$. By Theorem 3.18, any t columns of the array are linearly independent, so there cannot be a codeword of weight t or less in C^\perp . If the array does not have strength $t + 1$, some $t + 1$ columns are dependent, and so there is a codeword of weight $t + 1$ in the dual code, hence $d^\perp = t + 1$ [19, p. 66]. ■

The concept of dual codes is important in the study of orthogonal arrays. As seen above, it allowed us to establish the connection between orthogonal arrays and linear codes in the proof of Theorem 3.20. Moreover, since orthogonal arrays can be created using linear codes and linear codes are either self-dual or give rise to dual codes, this frequently results in connections between pairs of orthogonal arrays. For example, the codewords of the $(7, 8, 4)_2$ code, C , from Example 3.17 form a $OA(8, 7, 2, 2)$ orthogonal array, while the code words of its dual code, C^\perp , $(7, 16, 3)_2$, creates a $OA(16, 7, 2, 3)$ orthogonal array. We shall later extend the concept of duality to correlation immune generalized Boolean functions which were created using orthogonal arrays. Having covered a sufficient amount of background information, we are now in a position to introduce our construction method for higher order correlation immune generalized Boolean functions. To motivate the technique, we begin again by considering the “folklore” construction.

Consider a $CI(1)$ function $f \in \mathcal{GB}_5^q$, which was created using the folklore construc-

tion. Here the linear structure, $\mathbf{a} = 11111$, is used to partition \mathbb{V}_5 and for all $\mathbf{x} \in \mathbb{V}_5$, $f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{a})$. In particular, the set of input vectors $\{00000, 11111\}$ are mapped to the same output value. These two vectors constitute the $(5, 2, 5)_2$ linear code, and by Theorem 3.20, they also form the $OA(2, 5, 2, 1)$ orthogonal array. Viewed from an orthogonal array perspective, the folklore construction is carried out as follows: Let $G = (\mathbb{V}_5, \oplus)$ represent the abelian group of binary input vectors formed under the \oplus operation. $OA(2, 5, 2, 1)$ is a linear orthogonal array since it was created using the linear code $(5, 2, 5)_2$. Since $(5, 2, 5)_2$ is a linear code, it forms a subgroup of G . Let $O_0 = OA(2, 5, 2, 1)$. For each lexicographic ordered input vector \mathbf{x} from 00001 to 01111 we form the cosets, O_i , $1 \leq i \leq 15$, of O_0 by adding \mathbf{x} to the each of the two row vectors in O_0 . Then $\cup_{i=0}^{15} O_i = \mathbb{V}_5$ and we have partitioned \mathbb{V}_5 into 16 pairs of vectors. Moreover, according to Lemma 3.10, each of the cosets of O_0 is also an (order 1) orthogonal array. Therefore, by mapping the two row vectors within each orthogonal array, O_i , to the same output value, $c \in \mathbb{Z}_q$, we have constructed a CI(1) function.

The benefit of this construction method is that it allows us to use any linear orthogonal array $(m, n, 2, t)$, where $m = 2^\ell$ and $n > \ell$, to build a correlation immune (order t) generalized Boolean function $f \in \mathcal{GB}_n^q$, where $q = 2^{n-\ell}$.

Algorithm 4 $CI(t)$ generalized Boolean function construction

- 1: Select a linear orthogonal array $A = OA(m, n, 2, t)$, where $m = 2^\ell$ and $n > \ell$.
 - 2: **for** $k = 1$ to m **do**
 - 3: Add row vector $\mathbf{x}_k \in A$ to the set O_0 .
 - 4: **end for**
 - 5: Add O_0 to the set, S , of orthogonal arrays.
 - 6: **for** $j = 1$ to $2^{n-\ell} - 1$ **do**
 - 7: Select a vector $\mathbf{a}_j \in \mathbb{V}_n$, such that $\forall O_k \in S$, where $k < j$, $\mathbf{a}_j \notin O_k$.
 - 8: **for** $i = 1$ to m **do**
 - 9: Compute $\mathbf{y}_i = \mathbf{x}_i \oplus \mathbf{a}_j$, where \mathbf{x}_i are row vectors in O_0 .
 - 10: Add \mathbf{y}_i to O_j .
 - 11: **end for**
 - 12: Add O_j to S .
 - 13: **end for**
 - 14: Select a permutation, p , of the set $\{1, 2, \dots, n\}$
 - 15: **for** $i = 1$ to $2^{n-\ell}$ **do**
 - 16: Reorder the columns, \mathbf{c}_k , $k = 1$ to n , of O_i such that $O_i^{(p)} = \mathbf{c}_{p_1}, \mathbf{c}_{p_2}, \dots, \mathbf{c}_{p_n}$, where p_n is the n^{th} element of p .
 - 17: **end for**
 - 18: **for** $h = 1$ to $2^{n-\ell}$ **do**
 - 19: Select an output value $c_h \in \mathbb{Z}_q$, $2 \leq q \leq 2^{n-\ell}$.
 - 20: **for** $i = 1$ to m **do**
 - 21: Save the ordered pair, $\{\mathbf{x}_i, c_h\}$, where $\mathbf{x}_i \in O_h^{(p)}$, to a 2D array, f .
 - 22: **end for**
 - 23: **end for**
 - 24: Sort f so that the first elements of each ordered pair, $\{\mathbf{x}_i, c_h\} \in f$, appear in lexicographic order.
-

Proof of Correctness of the $CI(t)$ Generalized Boolean Function Construction: Suppose we wish to create a correlation immune (order t) generalized Boolean function $f \in \mathcal{GB}_n^q$ using the above described algorithm. We first select a suitable linear orthogonal array, $O_0 = OA(m, n, 2, t)$, such that t satisfies the desired correlation immunity order and n satisfies the required input variable length for our function. Since O_0 is a linear orthogonal array, its row vectors form a subgroup of \mathbb{V}_n . Let $m = 2^\ell$. By selecting an orthogonal array with m such that $2^{n-\ell} \geq q$ we ensure that our construction can achieve the requisite number of functional output values, q . Moreover, the fact that O_0 is simple and forms a subgroup of \mathbb{V}_n guarantees that O_0 along with its $2^{n-\ell} - 1$ cosets cover \mathbb{V}_n . We construct each coset O_i , $i = 1$ to $2^{n-\ell} - 1$, by selecting a vector $\mathbf{a} \in \mathbb{V}_n$ not present in O_0 (or any other coset). Lemma 3.10 tells us that each of these cosets is also an $OA(m, n, 2, t)$ orthogonal array. Having done so, we have thus partitioned \mathbb{V}_n into $2^{n-\ell}$ orthogonal arrays each of strength t . We now select one of the $n!$ possible permutations, $p = \{p_1, p_2, \dots, p_n\}$, of the integers $\{1, 2, \dots, n\}$, where p_n is the n^{th} element of the set p . Let $O_i = [\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n]$, where \mathbf{c}_j , $1 \leq j \leq n$, represents a column vector. We reorder the columns of each orthogonal array O_i such that $O_i^{(p)} = [\mathbf{c}_{p_1}, \mathbf{c}_{p_2}, \dots, \mathbf{c}_{p_n}]$. Since by Definition 3.8, each O_i , $i = 0$ to $2^{n-\ell} - 1$, must contain all 2^t possible row vectors for any combination of t columns, each resultant array $O_i^{(p)}$ will remain an orthogonal array. Moreover, while the column reordering will alter the vectors which occur within each orthogonal array $O_i^{(p)}$, the set of all orthogonal arrays $O_i^{(p)}$, $i = 0$ to $2^{n-\ell} - 1$, will still cover \mathbb{V}_n . To recognize that this is indeed the case, consider the following: The set of simple orthogonal arrays $S = \{O_0, O_1, \dots, O_{2^{n-\ell}-1}\}$ covers \mathbb{V}_n . There are a total of 2^n row vectors in \mathbb{V}_n , each of which is unique. Since we respect the same reordering scheme, $O_i^{(p)} = [\mathbf{c}_{p_1}, \mathbf{c}_{p_2}, \dots, \mathbf{c}_{p_n}]$, for $i = 0$ to $2^{n-\ell} - 1$, it must be the case that each vector in $S^{(p)} = \{O_0^{(p)}, O_1^{(p)}, \dots, O_{2^{n-\ell}-1}^{(p)}\}$ is also unique. Since there are also 2^n row vectors in $S^{(p)}$, it must be the case that the set of modified orthogonal arrays $S^{(p)}$ also covers \mathbb{V}_n . Finally, to each set of input vectors, $O_i^{(p)}$ we associate an output value $c_i \in \mathbb{Z}_q$, where $q \leq 2^{n-\ell}$. Since each orthogonal array $O_i^{(p)}$ is strength t , we have thus created a $CI(t)$ generalized Boolean function $f \in \mathcal{GB}_n^q$.

To illustrate the algorithm further, we provide the following example:

Example 3.21. Suppose we wish to construct a higher order ($t > 1$) correlation immune generalized Boolean function $f \in \mathcal{GB}_5^4$. We begin by finding a linear orthogonal array

suitable for the task. In this case, $OA(8, 5, 2, 2)$ is a good candidate. Let $O_0 = OA(8, 5, 2, 2)$.

$$O_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0. \end{bmatrix}$$

Since $OA(8, 5, 2, 2)$ is a linear orthogonal array, O_0 's row vectors form a subgroup of \mathbb{V}_5 . We can therefore cover \mathbb{V}_5 by forming the 3 cosets of O_0 . To do so, we iteratively proceed as follows: For $i = 1$ to 3 we form O_i by selecting a vector, $\mathbf{a} \in \mathbb{V}_n$, which is not present in all preceding orthogonal array's, O_j , where $j < i$. Then for each row vector $\mathbf{x}_k \in O_0$, $k = 1$ to 8, we compute $\mathbf{y}_k = \mathbf{x}_k \oplus \mathbf{a}$ and add it to O_i . Doing so produces the cosets

$$O_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1, \end{bmatrix} \quad O_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0, \end{bmatrix} \quad O_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0. \end{bmatrix}$$

Lemma 3.10 ensures that these newly formed cosets are all $OA(8, 5, 2, 2)$ orthogonal arrays in their own right. We now select a permutation, p of the set $\{1, 2, \dots, 5\}$, say for example $p = \{2, 1, 3, 5, 4\}$. For each of the orthogonal arrays, O_i , $i = 0$ to 3, we rearrange the columns of O_i such that $O_i^{(p)} = [\mathbf{c}_{p(1)}, \mathbf{c}_{p(2)}, \mathbf{c}_{p(3)}, \mathbf{c}_{p(4)}, \mathbf{c}_{p(5)}] = [\mathbf{c}_2, \mathbf{c}_1, \mathbf{c}_3, \mathbf{c}_5, \mathbf{c}_4]$,

$$O_0^{(p)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0, \end{bmatrix} \quad O_1^{(p)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0, \end{bmatrix} \quad O_2^{(p)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1, \end{bmatrix} \quad O_3^{(p)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0. \end{bmatrix}$$

By subsequently assigning the same output value from \mathbb{Z}_4 to the vectors within each or-

thogonal array, say for example $\{O_0^{(p)} \rightarrow 0, O_1^{(p)} \rightarrow 1, O_2^{(p)} \rightarrow 2, O_3^{(p)} \rightarrow 3\}$, we create the $CI(2)$ generalized Boolean function depicted in Table 3.5:

Table 3.5: A $CI(2)$ generalized Boolean function $f \in \mathcal{GB}_5^4$

\mathbb{V}_5	a_0	a_0	$a_0 \oplus a_1$	f
00000	0	0	0	0
00001	0	1	1	2
00010	1	0	1	1
00011	1	1	0	3
00100	1	0	1	1
00101	1	1	0	3
00110	0	0	0	0
00111	0	1	1	2
01000	1	1	0	3
01001	1	0	1	1
01010	0	1	1	2
01011	0	0	0	0
01100	0	1	1	2
01101	0	0	0	0
01110	1	1	0	3
01111	1	0	1	1
10000	0	1	1	2
10001	0	0	0	0
10010	1	1	0	3
10011	1	0	1	1
10100	1	1	0	3
10101	1	0	1	1
10110	0	1	1	2
10111	0	0	0	0
11000	1	0	1	1
11001	1	1	0	3
11010	0	0	0	0
11011	0	1	1	2
11100	0	0	0	0
11101	0	1	1	2
11110	1	0	1	1
11111	1	1	0	3

Given the fact that Algorithm 4 makes use of column permutations when constructing gen-

eralized Boolean functions, it is of interest to investigate when such actions result in new orthogonal arrays and partitions of \mathbb{V}_n .

Definition 3.22. An orthogonal array whose set of row vectors remains invariant under the full symmetric group S_n of column permutations is called a *symmetric orthogonal array*.

Example 3.23. The orthogonal array $OA(4, 3, 2, 2)$:

$$O = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

is a symmetric orthogonal array, since the set of O 's row vectors remain invariant under the full symmetric group S_3 of column permutations.

Remark 3.24. Given an orthogonal array, $O = OA(m, n, 2, t)$, it is a relatively straightforward matter to check whether or not it is symmetric. Let H represent the set of Hamming weights of all m row vectors in O . In order for O to be a symmetric orthogonal array, for each Hamming weight, $h \in H$, O must contain all vectors, $\mathbf{x} \in \mathbb{V}_n$, such that $wt(\mathbf{x}) = h$.

Lemma 3.25. Given a symmetric linear orthogonal array $O = OA(2^{n-1}, n, 2, t)$, the remaining set of vectors $\mathbb{V}_n \setminus O$ also forms a symmetric orthogonal array.

Proof. Let O_0 be a symmetric linear orthogonal array $OA(2^{n-1}, n, 2, t)$. Since O_0 is a linear orthogonal array, the row vectors of O_0 form an order 2^{n-1} abelian subgroup, $O < (\mathbb{V}_n, \oplus)$. We select a vector $\mathbf{a} \in \mathbb{V}_n$ which is not present in O_0 and add it in turn to each row vector in O_0 . The resultant set of vectors, O_1 , is the coset of O_0 and $O_0 \cup O_1 = \mathbb{V}_n$. Moreover, according to Lemma 3.10, O_1 is also a $OA(2^{n-1}, n, 2, t)$ orthogonal array. Let H represent the set of Hamming weights of all row vectors in O_0 . Since O_0 is symmetric, it must be the case that for each, $h \in H$, O_0 contains all vectors, $\mathbf{x} \in \mathbb{V}_n$ such that $wt(\mathbf{x}) = h$. This in turn means that O_1 contains all vectors $\mathbf{y} \in \mathbb{V}_n$ such that $wt(\mathbf{y}) \in \mathbb{Z}_n \setminus H$, thus demonstrating that O_1 is also a symmetric orthogonal array. ■

Definition 3.26. A partition of \mathbb{V}_n which remains invariant under the full symmetric group S_n of column permutations is called a *symmetric partition*.

Remark 3.27. Lemma 3.25 demonstrates the fact that binary symmetric linear orthogonal arrays with n constraints and size $2^{n-\ell}$ give rise to symmetric partitions of \mathbb{V}_n . However, it is possible for a partition containing subsets, some of which are not symmetric, to nonetheless be symmetric. To illustrate the point, consider the following:

Example 3.28. Below we list the linear orthogonal array $O_0 = OA(2, 4, 2, 1)$ along with its 7 cosets:

$$\begin{array}{llll} O_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} & O_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} & O_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} & O_3 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \\ O_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} & O_5 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} & O_6 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} & O_7 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

While O_0 clearly is a symmetric linear orthogonal array, given that it remains invariant under all column permutations, each of its cosets are not. Despite this fact, the set of all orthogonal arrays, $P = \{O_0, O_1, \dots, O_7\}$, is nonetheless symmetric. The reason for this is that P forms a group under the set Σ of the $4!$ column permutations. For example, for one such column permutation $\sigma = 4123$

$$\sigma = \begin{pmatrix} O_0 & O_1 & O_2 & O_3 & O_4 & O_5 & O_6 & O_7 \\ O_0 & O_4 & O_1 & O_2 & O_3 & O_7 & O_6 & O_5 \end{pmatrix} = (O_1 O_4 O_3 O_2)(O_5 O_7).$$

Proposition 3.29. *The partition of \mathbb{V}_n used in the folklore CI(1) construction is symmetric.*

Proof. The folklore construction partitions \mathbb{V}_n into 2^{n-1} pairs of complementary vectors. Every column in each pair contains complementary bits. For each vector pair \mathbf{x} and $\bar{\mathbf{x}}$, any column permutation σ therefore produces a pair of complementary vectors \mathbf{x}' and $\bar{\mathbf{x}}'$. Since each vector in the partition is unique and σ is applied to all vector pairs, the permutation results in 2^{n-1} pairs of complementary vectors. ■

A nonsymmetric partition of \mathbb{V}_n gives rise to multiple partitions of \mathbb{V}_n under the set of column permutations. The exact number of resultant partitions depends upon the partition in question, but is bounded above by $n!$.

Theorem 3.30. *Let $O = OA(2^\ell, n, 2, t)$, $n > \ell$, be a linear orthogonal array, and let F represent the set of distinct correlation immune (order t) generalized Boolean functions*

$f \in \mathcal{GB}_n^q$, where $q = 2^{n-\ell}$. The number $|F|$ of distinct $CI(t)$ generalized Boolean functions that can be constructed using O and Algorithm 4 is bounded by:

$$(2^{n-\ell})^{2^{n-\ell}} \leq |F| \leq n!(2^{n-\ell})^{2^{n-\ell}}.$$

Proof. Let $O = OA(2^\ell, n, 2, t)$, $\ell < n - 1$, be a linear orthogonal array. If O is symmetric and gives rise to a symmetric partition P of \mathbb{V}_n , the set of partitions produced by column permutations is singular. Since O is a linear orthogonal array, O along with its $2^{n-\ell} - 1$ cosets (each of which also are $OA(2^\ell, n, 2, t)$ orthogonal arrays) therefore cover \mathbb{V}_n . In order to ensure correlation immunity (order t) we assign the same output value to all row vector within each of the $2^{n-\ell}$ orthogonal arrays. Assigning a unique value to each orthogonal array establishes the maximum size of the image of f , $|f(\mathbb{V}_n)| = 2^{n-\ell}$. For each of the $2^{n-\ell}$ orthogonal arrays in P there are $q = 2^{n-\ell}$ choices for the output value, which in turn establishes the stated lower bound of $(2^{n-\ell})^{2^{n-\ell}}$. If, on the other hand, the partition P is nonsymmetric, then the set of column permutations will produce several distinct partitions of \mathbb{V}_n . Consider the extreme case: Suppose O contains row vectors, each of which has unique Hamming weight and each column of O is also unique. In this case, each of the $n!$ column permutations of O would produce a unique orthogonal array $O^{(p)}$. Each of these is a linear orthogonal array, and thus along with its cosets gives rise to a unique partition of \mathbb{V}_n . Each of the $n!$ partitions contain $2^{n-\ell}$ orthogonal arrays, so the maximum size of the image of f is again $|f(\mathbb{V}_n)| = 2^{n-\ell}$. For each of the $2^{n-\ell}$ orthogonal arrays in a given partition, there are $q = 2^{n-\ell}$ choices for the output value. Hence, as before, for each partition there are $(2^{n-\ell})^{2^{n-\ell}}$ possible ways of assigning OA-output value pairs. Thus, the upper bound for the total number of $CI(t)$ generalized Boolean function we can construct with O and Algorithm 4 is bounded above by $n!(2^{n-\ell})^{2^{n-\ell}}$. ■

Given the construction method of Algorithm 4, the maximum number of output values which correlation immune generalized Boolean function $f \in \mathcal{GB}_n^q$ can achieve is $2^n/m$, where m is the size of the linear orthogonal array $OA(m, n, 2, t)$. We use this fact along with the Singleton bound to establish bounds on the size of the image of f .

Theorem 3.31 (Singleton bound for $CI(t)$ generalized Boolean functions). *Let $f \in \mathcal{GB}_n^q$ be a $CI(t)$ generalized Boolean function constructed using a linear orthogonal array*

$OA(m, n, 2, t)$ and Algorithm 4. Then the size of the image of f is bounded by

$$2^{d-1} \leq |f(\mathbb{V}_n)| \leq 2^{n-t},$$

where d is the minimum distance of the linear code associated with the linear orthogonal array.

Proof. Let O be the linear orthogonal array $OA(m, n, 2, t)$ which was used to construct f in accordance with Algorithm 4. Let C denote the linear code associated with O , let $|C|$ denote the number of codewords in C , and let d denote minimum distance for C . Then $m = |C|$. C is a linear code, therefore it is simple. From Theorem 4.20 [19, p. 79] we know that, for a set of vectors C of length n with minimal distance d and strength t

$$s^t \leq |C| \leq s^{n-d+1},$$

where s is the vector alphabet size and the right-hand side bound assumes that C is a simple code. Letting $s = 2$ we then have:

$$2^t \leq |C| \leq 2^{n-d+1}.$$

Algorithm 4 partitions V_n into subsets of size m , each of which is subsequently assigned an output value from \mathbb{Z}_q . The maximum size of the image of f is therefore $2^n/m = 2^n/|C|$. This number is largest when $|C|$ is smallest and vice versa. Therefore:

$$2^{n-(n-d+1)} \leq |f(\mathbb{V}_n)| \leq 2^{n-t},$$

which establishes the stated bounds on the cardinality of the image of f . ■

Proposition 3.32 ($CI(t)$ generalized Boolean functions duality). *Let O be an $OA(m, n, 2, t)$ linear orthogonal array and let C be its corresponding $(n, m, d)_2$ linear code. Let C^\perp be the dual code of C and let O^\perp represent the dual orthogonal array associated with C^\perp . Let F represent the set of correlation immune (order t) generalized Boolean functions that can be constructed using O and Algorithm 4. If n is odd, or if n is even and the Hamming weight of at least one of O 's row vectors is not divisible by 2, then there exists a set F^\perp of*

correlation immune generalized Boolean functions which can be constructed using O^\perp .

Proof. This is a direct consequence of Theorem 3.20 and the existence of dual codes. Binary linear (even or doubly-even) self-dual codes occur when n is even and the Hamming weight of each codeword is divisible by 2 or 4 respectively [26, p. 27]. By stipulating that either n be odd, or n be even and O contain at least one row vector which is divisible by 2, we ensure that C is not a self-dual code. This means that a distinct O^\perp linear orthogonal array exists which can in turn be used in conjunction with Algorithm 4 to generate F^\perp . ■

Proposition 3.33. *Let $u \geq 1$, $\ell \leq n - 1$ and $q \leq 2^{n-\ell}$. When constructing correlation immune functions using Algorithm 4, $CI(2u)$ functions $f \in \mathcal{GB}_n^q$ exist if and only if $CI(2u+1)$ functions $f \in \mathcal{GB}_{n+1}^q$ exist.*

Proof. This is a direct consequence of Theorem 2.24 by Hedayat, Sloane and Stufken [19, p. 28], which states: An $OA(m, n, 2, 2u)$ orthogonal array exists if and only if an $OA(2m, n, 2, 2u+1)$ orthogonal array exists. In the interest of brevity, we omit their proof here. The interested reader may refer to their work for the proof of this orthogonal array result. ■

There are many known linear orthogonal arrays which are suitable for constructing higher order correlation immune generalized Boolean functions $f \in \mathcal{GB}_n^q$ using the method outlined in Algorithm 4. Using [19] and [41] we have, for the benefit of the reader, compiled an (incomplete) list of function parameters, n , q and t , along with the parameters of corresponding known linear orthogonal arrays in Table 3.6. Additionally, several of these linear orthogonal arrays can be found in Appendix C.

Table 3.6: Some orthogonal arrays and associated generalized Boolean function parameters

n	$q \leq$	$CI(t)$	OA
5	4	2	$OA(8, 5, 2, 2)$
6	4	3	$OA(16, 6, 2, 3)$
7	16	2	$OA(8, 7, 2, 2)$
7	8	3	$OA(16, 7, 2, 3)$
8	16	3	$OA(16, 8, 2, 3)$
9	4	5	$OA(2^7, 9, 2, 5)$
12	4	7	$OA(2^{10}, 12, 2, 7)$
15	2^{11}	2	$OA(16, 15, 2, 2)$
15	2^8	3	$OA(2^7, 15, 2, 3)$
15	2^7	4	$OA(2^8, 15, 2, 4)$
16	2^{11}	3	$OA(32, 16, 2, 3)$
16	32	7	$OA(2^{11}, 16, 2, 7)$
18	8	9	$OA(2^{15}, 18, 2, 9)$
20	2^{11}	5	$OA(2^9, 20, 2, 5)$
24	2^{14}	5	$OA(2^{10}, 24, 2, 5)$
24	2^{12}	7	$OA(2^{12}, 24, 2, 7)$
31	2^{26}	2	$OA(32, 31, 2, 2)$
32	2^{26}	3	$OA(64, 32, 2, 3)$
32	2^{21}	5	$OA(2^{11}, 32, 2, 5)$
32	2^6	15	$OA(2^{26}, 32, 2, 15)$

3.4 New From Old Correlation Immune Generalized Boolean Functions

In his original paper [40], Siegenthaler provided a construction of a large class of correlation immune (order t) functions on $n + 1$ variables by concatenating the truth tables of two n variable correlation immune (order t) Boolean functions. This method, along with the proof of its correctness, can be found in Cusick and Stănică's book on Cryptographic Boolean functions [11, p. 74]. We extend here their theorem (4.20) so that it applies to generalized Boolean functions. Before doing so, it is however necessary for us to generalize

Lemma 4.2 (f) [11, p. 56], also contained in their aforementioned monograph.

Lemma 3.34. *Let $f \in \mathcal{G}\mathcal{B}_n^q$ be a generalized Boolean function and let $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{V}_n$ be an input vector of f . Let $\mathbf{y} = (x_{i(1)}, \dots, x_{i(t)})$ be made up of an arbitrary choice of t of the variables x_i , and let $\mathbf{y}_0 = (y_1, \dots, y_t)$ be any fixed binary t -vector. Let $wt(f|_c)$ denote the number of occurrences of c in the truth table of f . If f is correlation immune of order t , then for all \mathbf{y} and for each \mathbf{y}_0 , $Pr(f(\mathbf{x}) = c | \mathbf{y} = \mathbf{y}_0) = Pr(f(\mathbf{x}) = c) = \frac{wt(f|_c)}{2^n}$.*

Proof. Let $f \in \mathcal{G}\mathcal{B}_n^q$ be a correlation immune (order t) generalized Boolean function. Then, since f is correlation immune of order t , for all c we have

$$Pr(\mathbf{y} = \mathbf{y}_0 | f(\mathbf{x}) = c) = \frac{Pr(\mathbf{y} = \mathbf{y}_0 \cap f(\mathbf{x}) = c)}{Pr(f(\mathbf{x}) = c)} = \frac{1}{2^t} \implies Pr(\mathbf{y} = \mathbf{y}_0 \cap f(\mathbf{x}) = c) = \frac{Pr(f(\mathbf{x}) = c)}{2^t}$$

and

$$Pr(f(\mathbf{x}) = c | \mathbf{y} = \mathbf{y}_0) = \frac{Pr(f(\mathbf{x}) = c \cap \mathbf{y} = \mathbf{y}_0)}{Pr(\mathbf{y} = \mathbf{y}_0)} = \frac{Pr(f(\mathbf{x}) = c)}{2^t \cdot Pr(\mathbf{y} = \mathbf{y}_0)} = \frac{Pr(f(\mathbf{x}) = c)}{2^t \cdot 2^{-t}} = \frac{wt(f|_c)}{2^n}.$$

■

Theorem 3.35. *Let $\mathbf{x} = (x_1, \dots, x_n)$ and suppose that we have correlation immune (order t) generalized Boolean functions, $f_1, f_2 \in \mathcal{G}\mathcal{B}_n^q$, such that $\forall c \in \mathbb{V}_n, Pr(f_1(\mathbf{x}) = c) = Pr(f_2(\mathbf{x}) = c) = p$. Then the function f of $n+1$ variables defined by*

$$f(\mathbf{x}, x_{n+1}) = x_{n+1}f_1(\mathbf{x}) + (x_{n+1} \oplus 1)f_2(\mathbf{x}) \quad (3.2)$$

is also correlation immune of order t and satisfies $Pr(f(\mathbf{x}) = c) = p$.

Proof. Let $\mathbf{y} = (x_{i(1)}, \dots, x_{i(t)})$ be made up of an arbitrary choice of t of the variables, x_i , and let $\mathbf{y}_0 = (y_1, \dots, y_t)$ be any fixed binary t -vector. Then since f_1 and f_2 do not depend on x_{n+1} we have for either fixed choice of the bit b , and $i = 1$ or 2 ,

$$Pr(f_i = c | \mathbf{y} = \mathbf{y}_0, x_{n+1} = b) = Pr(f_i = c | \mathbf{y} = \mathbf{y}_0) = Pr(f_i = c), \quad (3.3)$$

where the second equality follows from our hypothesis that f_i is correlation immune of

order t and using Lemma 3.34 above. Now (3.2) and (3.3) imply

$$Pr(f = c | \mathbf{y} = \mathbf{y}_0, x_{n+1} = 1) = Pr(f_1 = p)$$

and

$$Pr(f = c | \mathbf{y} = \mathbf{y}_0, x_{n+1} = 0) = Pr(f_2 = p)$$

so we obtain

$$Pr(f = c | \mathbf{y} = \mathbf{y}_0, x_{n+1} = b) = Pr(f = c) = p.$$

This implies that the value of f is independent of the choice of any subset of t of the $n + 1$ input variables, so f is correlation immune of order at least t . ■

Example 3.36. Table 3.7 provides an example of generalized Boolean functions which was constructed using Theorem 3.35.

Table 3.7: A Siegenthaler constructed $CI(1)$ function $f \in \mathcal{GB}_4^4$

\mathbb{V}_4	a_0	a_1	f
0000	0	0	0
0001	1	1	3
0010	0	1	2
0011	1	0	1
0100	1	0	1
0101	0	1	2
0110	1	1	3
0111	0	0	0
1000	0	1	2
1001	1	0	1
1010	1	1	3
1011	0	0	0
1100	0	0	0
1101	1	1	3
1110	1	0	1
1111	0	1	2

In the above example we see how correlation immune (order 1) Boolean functions can be used to construct new correlation immune (order 1) generalized Boolean functions. Care

must however be taken to ensure that all stipulated requirements are satisfied by the two selected generalized Boolean functions before proceeding with the construction. To illustrate the point, consider the following example in Table 3.8:

Table 3.8: A correlation immune generalized Boolean function construction failure

\mathbb{V}_3	a_0	a_1	f
000	1	0	1
001	0	1	2
010	0	1	2
011	1	0	1
100	0	0	0
101	1	1	3
110	1	1	3
111	0	0	0

In this case, both Boolean functions a_0 and a_1 are $CI(1)$, yet the generalized Boolean function f fails to be correlation immune. The cause of the failure lies in the fact that, in order for the generalized Siegenthaler construction to work, Theorem 3.35 requires that the two generalized Boolean functions f_1 and f_2 are such that $\forall c \in f_1(\mathbb{V}_n) = f_2(\mathbb{V}_n)$, $Pr(f_1(\mathbf{x}) = c) = Pr(f_2(\mathbf{x}) = c) = p$. In this instance, $f_1(\mathbb{V}_n) = \{1, 2\} \neq \{0, 3\} = f_2(\mathbb{V}_n)$. This disagreement between the output values in the first and second half of the truth table of f results in the associated conditional probabilities not equaling the required values. For example, $Pr(x_1 = 1 | f(\mathbf{x}) = 3) = 1$.

3.5 Necessary and Sufficient Conditions for Correlation Immune Generalized Boolean Functions

Suppose, as depicted in Figure 3.1, we wish to design a q -ary sequence generator that uses k linear feedback shift registers (LFSRs) which in turn feed a generalized Boolean function $f \in \mathcal{GB}_n^q$, $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, where $a_j \in \mathcal{B}_n$.

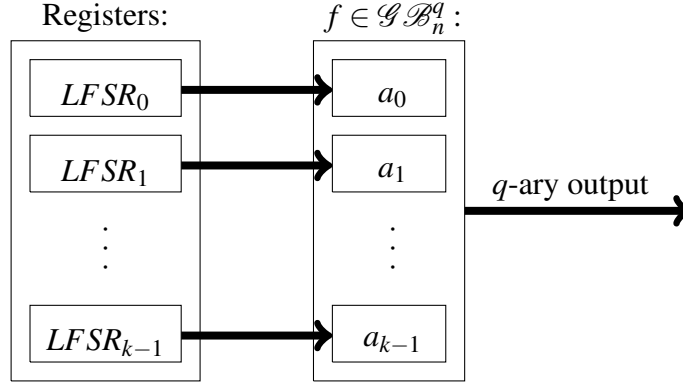


Figure 3.1: q -ary sequence generator

Suppose further that we wish to ensure that our function is immune to correlation attacks. Considering the problem for a moment, we quickly recognize that the q -ary nature of the output sequence does not provide any additional security. By binary expansion of each of the output values in the sequence, an attacker could simply employ a divide-and-conquer approach and perform k separate correlation attacks, one on each of our function's k constituent Boolean functions a_j . Clearly in order for a generalized Boolean function $f \in \mathcal{G} \mathcal{B}_n^q$ used in this manner to be considered correlation immune, the governing CI criteria must be satisfied by each of the constituent Boolean functions a_j , $0 \leq j \leq k-1$.

Lemma 3.37. *Let $f \in \mathcal{G} \mathcal{B}_n^q$ be a correlation immune (order t) function and let \mathbb{V}_n represent the set of binary input vectors, $\mathbf{x} = (x_n, \dots, x_1)$. Let $c \in f(\mathbb{V}_n)$ be an output value of f and $V_c = \{\mathbf{x} \in \mathbb{V}_n : f(\mathbf{x}) = c\}$. Let $\mathbf{y} = (x_{i(1)}, \dots, x_{i(t)})$ be an arbitrary choice of t of the variables, x_i , and let $\mathbf{y}_0 = (y_1, \dots, y_t)$ be any fixed binary t -vector. Assume that there exists a partition $V_c = \cap_{i=1}^r W_i$, $W_i \cup W_j = \emptyset$, $i \neq j$, and for all $W \in \{W_1, \dots, W_r\}$ and for each \mathbf{y}_0 , $\Pr(\mathbf{y} = \mathbf{y}_0 \mid f|_W = c) = 2^{-t}$. Then for all $U = \cup_{i \in \{1, 2, \dots, r\}} W_i$, $\Pr(\mathbf{y} = \mathbf{y}_0 \mid f|_U = c) = 2^{-t}$, for each \mathbf{y}_0 .*

Proof. Let $f \in \mathcal{G} \mathcal{B}_n^q$ be a correlation immune (order t) function. Let $c \in f(\mathbb{V}_n)$ be an output value of f and let $V_c = \{\mathbf{x} \in \mathbb{V}_n : f(\mathbf{x}) = c\}$. Let $\{W_1, \dots, W_r\}$ be mutually disjoint sets which partition V_c such that for every $W_i \in \{W_1, \dots, W_r\}$ and $\forall \mathbf{y}$ and each \mathbf{y}_0 : $\Pr(\mathbf{y} = \mathbf{y}_0 \mid f|_{W_i}) = 2^{-t}$. Without loss of generality, let U be an arbitrary union of s sets chosen

from $\{W_1, \dots, W_r\}$, where $2 \leq s \leq r$. Since for each W_i involved in U and $\forall \mathbf{y}$ and each \mathbf{y}_0 : $Pr(\mathbf{y} = \mathbf{y}_0 | f|_{W_i} = c) = 2^{-t}$, it must be the case that there are $2^{-t}|W_i|$ vectors $\mathbf{x} \in W_i$, which satisfy each condition $\mathbf{y} = \mathbf{y}_0$ for each subset W_i . The subsets W_i are disjoint, therefore, the total number of vectors which satisfy each specific condition, $\mathbf{y} = \mathbf{y}_0$, is

$$\frac{|W_1|}{2^t} + \frac{|W_2|}{2^t} + \dots + \frac{|W_s|}{2^t} = \frac{1}{2^t} \sum_{i=1}^s |W_i| = \frac{1}{2^t} |U|.$$

This in turn means that $Pr(\mathbf{y} = \mathbf{y}_0 | f|_U = c) = 2^{-t}$, regardless of our choice of U . \blacksquare

Theorem 3.38. *If f is a correlation immune (order t) generalized Boolean function, then all of its constituent Boolean functions are also correlation immune (order t).*

Proof. Let $\mathbf{x} \in \mathbb{V}_n$ and let $f \in \mathcal{GB}_n^q$ be a correlation immune (order t) generalized Boolean function, where $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, $a_j \in \mathcal{B}_n$. Suppose $c \in f(\mathbb{V}_n)$. Let $c_2(j)$ represent the j^{th} bit of the binary expansion of c , such that $f(\mathbf{x}) = c$ and $a_j(\mathbf{x}) = c_2(j)$. Since, $c_2(j) \in \mathbb{F}_2$, for each function a_j , the binary expansion of the elements of $f(\mathbb{V}_n)$ partition \mathbb{V}_n into disjoint sets $V_{0(1)}, V_{0(2)}, \dots, V_{0(r)}$ and $V_{1(1)}, V_{1(2)}, \dots, V_{1(s)}$, such that $r + s = |f(\mathbb{V}_n)|$ and for all $\mathbf{x} \in V_{0(\gamma)}$, where $1 \leq \gamma \leq r$, $a_j(\mathbf{x}) = 0$ and for all $\mathbf{x} \in V_{1(\delta)}$, where $1 \leq \delta \leq s$, $a_j(\mathbf{x}) = 1$. Let $\mathbf{y} = (x_{i(1)}, \dots, x_{i(t)})$ be made up of an arbitrary choice of t of the variables, x_i , and let $\mathbf{y}_0 = (y_1, \dots, y_t)$ be any fixed binary t -vector. Then, since f is correlation immune (order t), for each \mathbf{y}_0 and for every $c \in f(\mathbb{V}_n)$, we know that $Pr(\mathbf{y} = \mathbf{y}_0 | f(\mathbf{x}) = c) = 2^{-t}$. This in turn means that for each $V_{0(\gamma)}$ and each $V_{1(\delta)}$: $Pr(\mathbf{y} = \mathbf{y}_0 | f|_{V_{0(\gamma)}}(\mathbf{x}) = 0) = Pr(\mathbf{y} = \mathbf{y}_0 | f|_{V_{0(\delta)}}(\mathbf{x}) = 0) = 2^{-t}$. Turning our attention to the Boolean function, a_j , this implies that for each \mathbf{y}_0 and every $V_{0(\gamma)}$ and $V_{1(\delta)}$: $Pr(\mathbf{y} = \mathbf{y}_0 | a_j|_{V_{0(\gamma)}}(\mathbf{x}) = 0) = Pr(\mathbf{y} = \mathbf{y}_0 | a_j|_{V_{0(\delta)}}(\mathbf{x}) = 0) = 2^{-t}$. This can be viewed as a relabeling of f 's outputs from c to $c_2(j)$. If it were not possible to succeed in doing so, it would mean that f failed to be CI(t) for one or more of its output values c . Given this partitioning of a_j into individually CI(t) components, we let $V_0 = \cup_{\eta=1}^r V_{0(\eta)}$ and $V_1 = \cup_{\tau=1}^s V_{1(\tau)}$ and apply Lemma 3.37 which tells us that for each \mathbf{y}_0 , $Pr(\mathbf{y} = \mathbf{y}_0 | a_j|_{V_0}(\mathbf{x}) = 0) = Pr(\mathbf{y} = \mathbf{y}_0 | a_j|_{V_1}(\mathbf{x}) = 1) = 2^{-t}$, thus demonstrating that for all j , $0 \leq j \leq k-1$, a_j is a correlation immune (order t) Boolean function. \blacksquare

Theorem 3.38 guarantees that generalized Boolean functions which are correlation immune are not susceptible to binary output decomposition followed by correlation attacks

carried out on its Boolean function components. However, since cryptographers may wish to construct correlation immune generalized Boolean functions using correlation immune Boolean function as building blocks, we would also like to establish the criteria under which such functions ensure the resultant generalized Boolean function is correlation immune. As previously observed in Table 3.8, the fact that a generalized Boolean function f has Boolean functional components, all of which are correlation immune, is not sufficient to ensure that f itself is correlation immune.

Lemma 3.39. *Let X and Y be rectangular arrays, each containing m rows of binary vectors of length n .*

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & & & \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & & & \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}$$

Let \mathbf{x}_j and \mathbf{y}_j represent the j^{th} column vector of each respective array. Then, X and Y contain identical multisets of row vectors if and only if, for all j , $1 \leq j \leq n$, $wt(\mathbf{x}_j) = wt(\mathbf{y}_j)$ and the pairwise distances between column vectors, $d(\mathbf{x}_j, \mathbf{x}_k) = d(\mathbf{y}_j, \mathbf{y}_k)$ for all combinations j, k , where $1 \leq j, k \leq n$.

Proof. (\Rightarrow) Let X and Y be rectangular arrays each of which contain m rows of binary vectors of length n . Let the row vectors of X and Y be exhaustively constructed using identical multisets of size m . Let \mathbf{x}_j and \mathbf{y}_j represent the j^{th} column vector of each respective array. For each array, there are $m!$ orderings of the row vectors. Without loss of generality, select one such ordering for X and one ordering for Y . Now, X and Y were exhaustively constructed using row vectors taken from identical multisets, so despite any possible different orderings, for all j , $1 \leq j \leq n$, $wt(\mathbf{x}_j) = wt(\mathbf{y}_j)$. For each array, X and Y , we now create $\binom{n}{2}$ sub-arrays $X_{(j,k)}$ and $Y_{(j,k)}$ where each row i , from 1 to m , has elements (x_{ij}, x_{ik}) or (y_{ij}, y_{ik}) , respectively. Since X and Y have row vectors taken from identical multisets of size m , for each possible combination j and k , $1 \leq j, k \leq n$, it must also be the case that each sub-array $X_{(j,k)}$ and $Y_{(j,k)}$ form identical multisets of two element row vectors. In order for $d(\mathbf{x}_j, \mathbf{x}_k) \neq d(\mathbf{y}_j, \mathbf{y}_k)$ it would mean that $X_{(j,k)}$ and $Y_{(j,k)}$ had different multisets of

two-element row vectors. Since this is not the case, we conclude that $d(\mathbf{x}_j, \mathbf{x}_k) = d(\mathbf{y}_j, \mathbf{y}_k)$ for all combinations j, k , where $1 \leq j, k \leq n$.

(\Leftarrow) Let X and Y be two rectangular arrays each containing m rows of binary vectors of length n . Let \mathbf{x}_j and \mathbf{y}_j represent the j^{th} column vector of each respective array, and let X and Y be such that for all j , $1 \leq j \leq n$, $wt(\mathbf{x}_j) = wt(\mathbf{y}_j)$ and for all combinations of columns j, k , where $1 \leq j, k \leq n$, $d(\mathbf{x}_j, \mathbf{x}_k) = d(\mathbf{y}_j, \mathbf{y}_k)$. For each array, create $\binom{n}{2}$ sub-arrays $X_{(j,k)}$ and $Y_{(j,k)}$ where each row i , from 1 to m , has elements (x_{ij}, x_{ik}) or (y_{ij}, y_{ik}) respectively. To each sub-array, $X_{(j,k)}$ and $Y_{(j,k)}$ associate the 3-tuple $(wt(\mathbf{x}_j), wt(\mathbf{x}_k), d(\mathbf{x}_j, \mathbf{x}_k))$ or $(wt(\mathbf{y}_j), wt(\mathbf{y}_k), d(\mathbf{y}_j, \mathbf{y}_k))$, respectively. Now, $d(\mathbf{x}_j, \mathbf{x}_k) = \sum_{i=1}^m x_{ij} \oplus x_{ik}$ and $d(\mathbf{y}_j, \mathbf{y}_k) = \sum_{i=1}^m y_{ij} \oplus y_{ik}$. Therefore, the parity of the bits in each specific column of row i differ for each bit combination $(1 \oplus 0$ and $0 \oplus 1)$ which contributes to the cumulative distance between the column vectors. This is also the case for bit combinations $(1 \oplus 1$ and $0 \oplus 0)$ which do not contribute to the cumulative distance. Consequently, it is not possible to obtain two similar distance values between column vectors using different bit combinations, without altering the respective column weights. Our 3-tuples $(wt(\mathbf{x}_j), wt(\mathbf{x}_k), d(\mathbf{x}_j, \mathbf{x}_k))$ and $(wt(\mathbf{y}_j), wt(\mathbf{y}_k), d(\mathbf{y}_j, \mathbf{y}_k))$ are therefore unique irrespective of row vector order. Since $wt(\mathbf{x}_j) = wt(\mathbf{y}_j)$ for all i , $1 \leq i \leq n$ and $d(\mathbf{x}_j, \mathbf{x}_k) = d(\mathbf{y}_j, \mathbf{y}_k)$ for all combinations of columns j, k , where $1 \leq j, k \leq n$, it must be the case that $(wt(\mathbf{x}_j), wt(\mathbf{x}_k), d(\mathbf{x}_j, \mathbf{x}_k))$ and $(wt(\mathbf{y}_j), wt(\mathbf{y}_k), d(\mathbf{y}_j, \mathbf{y}_k))$ agree for all $X_{(j,k)}$ and $Y_{(j,k)}$. We have thus shown that X and Y must contain the same multisets of row vectors. \blacksquare

Theorem 3.40. *Let $f = f_1 \parallel f_2$ be a generalized Boolean function created using the generalized Siegenthaler construction in Theorem 3.35, such that $f \in \mathcal{GB}_{n+1}^q$, $f_1, f_2 \in \mathcal{GB}_n^q$, and f_1 and f_2 are both correlation immune (order t) functions. Let $f_1(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$ and $f_2(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j b_j(\mathbf{x})$, where $a_j, b_j \in \mathcal{B}_n$ and $\mathbf{x} \in \mathbb{V}_n$. Then f is correlation immune (order t) if and only if for all j and h , $0 \leq j, h \leq k-1$, the Boolean functions \mathbf{a}_j and \mathbf{b}_j are such that $wt(\mathbf{a}_j) = wt(\mathbf{b}_j)$ and the pairwise distances $d(\mathbf{a}_j, \mathbf{a}_h) = d(\mathbf{b}_j, \mathbf{b}_h)$.*

Proof. (\Rightarrow) Let $f \in \mathcal{GB}_{n+1}^q$ be a generalized Boolean function created by concatenating two $CI(t)$ generalized Boolean functions $f_1, f_2 \in \mathcal{GB}_n^q$ in accordance with Theorem 3.35. The function f is correlation immune (order t), so it must be the case that for all $\mathbf{x} \in \mathbb{V}_n$ and all output values $c \in \mathbb{Z}_q$, $c \in f_1(\mathbb{V}_n) \cap f_2(\mathbb{V}_n)$, and $Pr(f_1(\mathbf{x}) = c) = Pr(f_2(\mathbf{x}) = c)$. Let

$c_2(j)$ represent the j^{th} bit of the binary expansion of c , such that for $0 \leq j \leq k-1$, $f_1(\mathbf{x}) = c$ and $a_j(\mathbf{x}) = c_2(j)$ and $f_2(\mathbf{z}) = c$ and $b_j(\mathbf{z}) = c_2(j)$. Consider the two $2^n \times k$ arrays of truth table values for \mathbf{a}_j and \mathbf{b}_j :

\mathbf{a}_0	\mathbf{a}_1	\cdots	\mathbf{a}_{k-1}	\mathbf{f}_1
$a_{1,0}$	$a_{1,1}$	\cdots	$a_{1,k-1}$	$f_{1,1}$
$a_{2,0}$	$a_{2,1}$	\cdots	$a_{2,k-1}$	$f_{2,1}$
$a_{3,0}$	$a_{3,1}$	\cdots	$a_{3,k-1}$	$f_{3,1}$
\vdots	\vdots		\vdots	\vdots
$a_{2^n,0}$	$a_{2^n,1}$	\cdots	$a_{2^n,k-1}$	$f_{2^n,1}$

\mathbf{b}_0	\mathbf{b}_1	\cdots	\mathbf{b}_{k-1}	\mathbf{f}_2
$b_{1,0}$	$b_{1,1}$	\cdots	$b_{1,k-1}$	$f_{1,2}$
$b_{2,0}$	$b_{2,1}$	\cdots	$b_{2,k-1}$	$f_{2,2}$
$b_{3,0}$	$b_{3,1}$	\cdots	$b_{3,k-1}$	$f_{3,2}$
\vdots	\vdots		\vdots	\vdots
$b_{2^n,0}$	$b_{2^n,1}$	\cdots	$b_{2^n,k-1}$	$f_{2^n,2}$

Since the probabilities of c occurring in the two functions must be equal, the number of instances of c in \mathbf{f}_1 and \mathbf{f}_2 must be the same. This in turn means that the number of instances of c_2 occurring as a row vector must be the same for both arrays. By Lemma 3.39 the two arrays are such that for all j and h , $1 \leq j, h \leq k-1$, $wt(\mathbf{a}_j) = wt(\mathbf{b}_j)$, and all pairwise distances $d(\mathbf{a}_j, \mathbf{a}_h) = d(\mathbf{b}_j, \mathbf{b}_h)$.

(\Leftarrow) Let f_1 and f_2 be two n -variable correlation immune (order t) generalized Boolean functions. Let $f_1(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, and $f_2(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j b_j(\mathbf{x})$, where $a_j, b_j \in \mathcal{B}_n$ and $\mathbf{x} \in \mathbb{V}_n$. For all j and h , where $0 \leq j, h \leq k-1$, let the Boolean function truth tables be such that $wt(\mathbf{a}_j) = wt(\mathbf{b}_j)$ and $d(\mathbf{a}_j, \mathbf{a}_h) = d(\mathbf{b}_j, \mathbf{b}_h)$. This ensures that each function's $2^n \times k$ array of Boolean values contain the same multisets of binary row vectors. For each k -long binary vector c_2 in each multisets, there exists a corresponding value $c \in \mathbb{Z}_q$ in respective truth tables of \mathbf{f}_1 and \mathbf{f}_2 . Thus if the frequency of each distinct binary row vector agrees between the two multisets, so too does the frequency of each value c in \mathbf{f}_1 and \mathbf{f}_2 . We therefore conclude that for all c , $Pr(f_1(\mathbf{x}) = c) = Pr(f_2(\mathbf{x}) = c)$. Moreover, since f_1 and f_2 also agree with respect to dimension and correlation immunity order, we satisfy the requisite preconditions under which the generalized Siegenthaler construction may be used. Carrying out the construction we thus create the generalized Boolean function $f = f_1 \parallel f_2$, where $f \in \mathcal{GB}_{n+1}^q$. According to Theorem 3.35 this function is correlation immune of order t . ■

Theorem 3.41. *Let $f \in \mathcal{GB}_n^q$ be the generalized Boolean function $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, where $0 \leq j \leq k-1$, $a_j \in \mathcal{B}_n$ and $\mathbf{x} \in \mathbb{V}_n$. Then f is correlation immune (order t) if and only if all Boolean functions a_j are $CI(t)$ and use the same partition P of \mathbb{V}_n consisting of q orthogonal arrays, O_j , each of strength t .*

Proof. (\Rightarrow) Let f and the functions a_j be as described. Let P be a partition of \mathbb{V}_n consisting of q orthogonal arrays, O_h , $0 \leq h \leq q-1$, each of strength t . Suppose that for all $O_h \in P$ each function a_j uses the partition P . Then for each h and all vectors, $\mathbf{x} \in O_h$, $(a_0(\mathbf{x}), a_1(\mathbf{x}), \dots, a_{k-1}(\mathbf{x}))$ is a unique binary vector c_2 , and $a_0(\mathbf{x}) + 2a_1(\mathbf{x}) + \dots + 2^{k-1}a_{k-1}(\mathbf{x}) = c \in \mathbb{Z}_q$ is thus a unique output value for f . Consequently, f is correlation immune (order t).

(\Leftarrow) Let $f \in \mathcal{GB}_n^q$ be a correlation immune (order t) generalized Boolean function. Then according to Theorem 3.9, associated with f there is a partition P consisting of q strength t orthogonal arrays, O_h , $0 \leq h \leq q-1$, such that for each distinct output value $c_h \in f(\mathbb{V}_n)$, there exist an O_h such that $O_h = \{\mathbf{x} \in O_h : f(\mathbf{x}) = c_h\}$. Since $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, this means that for each c and all Boolean functions, a_j , there must exist an O_h such that $O_h = \{\mathbf{x} \in O_h : (a_0(\mathbf{x}), 2a_1(\mathbf{x}), \dots, 2^{k-1}a_{k-1}(\mathbf{x})) = c_h\}$. This in turn means that each Boolean function a_j utilizes the partition P . Moreover, by applying Lemma 3.37 to P and each respective Boolean function a_j , we conclude that a_j is $CI(t)$. \blacksquare

3.6 Correlation Immunity and the Walsh-Hadamard Transform

The Walsh transform is a very useful tool when studying Boolean functions. Cusick and Stănică provide the following lemma regarding correlation immunity of order t in their book on Cryptographic Boolean Functions and Applications [11]:

Lemma 3.42. [11, p. 56]

A [Boolean] function $f(\mathbf{x})$ in n variables is correlation immune of order t , $1 \leq t \leq n$, if and only if all of the Walsh transforms

$$\mathcal{W}_f(\mathbf{w}) = \sum_{\mathbf{x} \in \mathbb{V}_n} (-1)^{f(\mathbf{x}) \oplus \mathbf{x} \cdot \mathbf{w}} = 0, \quad 1 \leq wt(\mathbf{w}) \leq t.$$

It is certainly possible to define a correlation immunity notion based on the Walsh-Hadamard transform for generalized Boolean functions. To this end, we say that a generalized Boolean function $f \in \mathcal{GB}_n^q$ is *generalized correlation immune of order t* , denoted $gCI(t)$, if and only if $\mathcal{H}_f(\mathbf{w}) = 0$, where for all \mathbf{w} , with $1 \leq wt(\mathbf{w}) \leq t$. One naturally wonders whether or not this concept is equivalent to the probabilistic paradigm under which we have thus far been operating. We demonstrate in fact, that a function that is $CI(1)$, is also $gCI(1)$, but the converse is in general not true. For simplicity's sake, we consider here only the case when $t = 1$. The basic approach taken in the theorem that follows can however also be used to prove the cases when $t > 1$.

Theorem 3.43. *Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function. If f is $CI(1)$, then f is $gCI(1)$.*

Proof. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function and let $\mathbf{w} \in \mathbb{V}_n$ and $wt(\mathbf{w}) = 1$. That is, $\mathbf{w} = (0, \dots, 0, \overset{i}{\underset{\downarrow}{1}}, 0, \dots, 0)$, for some i . Now,

$$\begin{aligned}
\mathcal{H}_f(\mathbf{w}) &= \sum_{\mathbf{x}} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{w} \cdot \mathbf{x}} \\
&= \sum_{\mathbf{x}, x_i=0} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{w} \cdot \mathbf{x}} + \sum_{\mathbf{x}, x_i=1} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{w} \cdot \mathbf{x}} \\
&= \sum_{\substack{c=0 \\ \mathbf{x}, f(\mathbf{x})=c \\ x_i=0}}^{q-1} \zeta^c (-1)^{\mathbf{w} \cdot \mathbf{x}} + \sum_{\substack{c=0 \\ \mathbf{x}, f(\mathbf{x})=c \\ x_i=1}}^{q-1} \zeta^c (-1)^{\mathbf{w} \cdot \mathbf{x}} \\
&= \sum_{\substack{c=0 \\ \mathbf{x}, f(\mathbf{x})=c \\ x_i=0}}^{q-1} \zeta^c - \sum_{\substack{c=0 \\ \mathbf{x}, f(\mathbf{x})=c \\ x_i=1}}^{q-1} \zeta^c \\
&= \sum_{c=0}^{q-1} \eta_{0c} \zeta^c - \sum_{c=0}^{q-1} \eta_{1c} \zeta^c = \sum_{c=0}^{q-1} (\eta_{0c} - \eta_{1c}) \zeta^c,
\end{aligned}$$

where $\eta_{0c} = |\{x | f(\mathbf{x}) = c, x_i = 0\}|$ and $\eta_{1c} = |\{x | f(\mathbf{x}) = c, x_i = 1\}|$. Since f is $CI(1)$, $\eta_{0c} = \eta_{1c}$ for all c , therefore $\mathcal{H}_f(\mathbf{w}) = 0$. ■

Unfortunately, as we previously discovered when exploring balancedness in Chapter 2, things in the generalized setting have a tendency of becoming a bit more complicated than

that which one experiences in the classical Boolean environment. Such is also the case when it comes to correlation immunity. While the probabilistic point of view we have thus far been operating under is consistent with the correlation immunity notion using the generalized Walsh-Hadamard transform, the converse is in general not true. To see that this is indeed the case, consider the following generalized Boolean function $f \in \mathcal{GB}_4^4$.

Table 3.9: A *non* – $CI(1)$ function $f \in \mathcal{GB}_4^4$, where $H_f(\mathbf{w}) = 0$

∇_4	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
f	0	0	0	2	0	2	2	0	2	0	1	3	3	1	0	0

The 4th root of unity is $\zeta_4 = i$. Letting $\mathbf{w} \in \{0001, 0010, 0100, 1000\}$, we compute $\mathcal{H}_f(\mathbf{w})$, which yields the following:

$$\mathcal{H}_f(0001) = i^0 + i^0 + i^0 + i^2 + i^2 + i^1 + i^3 + i^0 - i^0 - i^2 - i^2 - i^0 - i^0 - i^3 - i^1 - i^0 = 0,$$

$$\mathcal{H}_f(0010) = i^0 + i^0 + i^0 + i^2 + i^2 + i^0 + i^3 + i^1 - i^0 - i^2 - i^2 - i^0 - i^1 - i^3 - i^0 - i^0 = 0,$$

$$\mathcal{H}_f(0100) = i^0 + i^0 + i^0 + i^2 + i^2 + i^0 + i^1 + i^3 - i^0 - i^2 - i^2 - i^0 - i^3 - i^1 - i^0 - i^0 = 0,$$

and

$$\mathcal{H}_f(1000) = i^0 + i^0 + i^0 + i^2 + i^0 + i^2 + i^2 + i^0 - i^2 - i^0 - i^1 - i^3 - i^3 - i^1 - i^0 - i^0 = 0.$$

Since the generalized Walsh-Hadamard transform, $\mathcal{H}_f(\mathbf{w})$ equals 0 for each Hamming weight 1 vector \mathbf{w} , the function f is $gCI(1)$. However, by inspection, one quickly observes that f is not $CI(1)$. For example, the two occurrences of the output value 1 both occur in the second half of the truth table. Thus, when considering the most significant (lexicographically ordered) bit position $i = 4$, one must conclude that f cannot be $CI(1)$.

3.7 Rotation Symmetric Correlation Immune Generalized Boolean Functions

Having discussed several methods of constructing correlation immune generalized Boolean functions, we now turn our attention to correlation immune generalized Boolean functions, which are also rotation symmetric. Rotation symmetric Boolean functions were introduced by Pieprzyk and Qu in 1999 [33], though they appear in the work of Filiol and Fontaine [15]

as idempotents, the preceding year. These functions remain invariant under cyclic rotations of their input vectors, and are of particular importance as components of cryptographic hashing algorithms, where they reduce computational complexity by allowing reuse of results obtained in previous iterations of an algorithm. Building upon our previously developed foundation of orthogonal array aided constructions, we will in this section extend the approach and demonstrate a method for constructing correlation immune and rotation symmetric generalized Boolean functions. Before embarking on this endeavor, we cover the following requisite material.

We adopt Cusick and Stănică's notation and generalize the definition of rotation symmetric Boolean functions from [11, p.121].

Let $(x_1, x_2, \dots, x_n) \in \mathbb{V}_n$. For $1 \leq \kappa \leq n$ we define

$$\rho_n^\kappa(x_i) = \begin{cases} x_{i+\kappa} & \text{if } i + \kappa \leq n, \\ x_{i+\kappa-n} & \text{if } i + \kappa > n, \end{cases}$$

which naturally extends to vectors.

Definition 3.44. A generalized Boolean function f is *rotation symmetric* (RotS) if and only if for any $(x_1, x_2, \dots, x_n) \in \mathbb{V}_n$,

$$f(\rho_n^\kappa(x_1, \dots, x_n)) = f(x_1, \dots, x_n),$$

for any $1 \leq \kappa \leq n$.

Definition 3.45. [19, p. 88] A linear code is called *cyclic* if whenever

$$(c_0, c_1, \dots, c_{k-2}, c_{k-1}) \tag{3.4}$$

is a codeword, then so too is

$$(c_1, c_2, \dots, c_{k-1}, c_0) \tag{3.5}$$

(that is, codewords are invariant under cyclic rotations).

Definition 3.46. [19, p. 88] An orthogonal array O , denoted $\overleftarrow{OA}(m, n, 2, t)$, is *cyclic* if it is linear and whenever (3.4) is a row vector in O , then (3.5) is a row vector in O .

As is customary in coding theory, we concisely represent a set of cyclic vectors using a single vector in angled brackets. Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$, then for $\kappa = 1$ to n , $\langle \mathbf{x} \rangle = \rho_n^\kappa(x_1, x_2, \dots, x_n)$. For example: $\langle 0001 \rangle = \{1000, 0100, 0010, 0001\}$. Additionally, given a vector $\mathbf{x} \in \mathbb{V}_n$, we define its cyclic period, $p_{\mathbf{x}}$, where $1 \leq p_{\mathbf{x}} \leq n$, as $p_{\mathbf{x}} = |\langle \mathbf{x} \rangle|$.

Definition 3.47. A partition of \mathbb{V}_n which remains invariant under the set of column rotations $\rho_n^k(\mathbf{x}_i)$, $1 \leq k \leq n$, is called a *rotation symmetric partition*.

Several of the previously discussed linear codes and linear orthogonal arrays were cyclic. Our new construction of *RotS* and *CI(t)* generalized Boolean functions relies upon cyclic orthogonal arrays. To highlight our approach, we revisit a familiar orthogonal array.

Example 3.48. Suppose we wish to construct a *RotS* and *CI(1)* generalized Boolean function $f \in \mathcal{GB}_4^4$. We begin again with the linear orthogonal array $O_0 = OA(2, 4, 2, 1)$. As seen in Example 3.28, this orthogonal array is symmetric and thus must also be *RotS*. As before, we list O_0 along with its 7 cosets:

$$\begin{array}{llll} O_0 = \begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1, \end{array} & O_1 = \begin{array}{cccc} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0, \end{array} & O_2 = \begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1, \end{array} & O_3 = \begin{array}{cccc} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1, \end{array} \\ O_4 = \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1, \end{array} & O_5 = \begin{array}{cccc} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0, \end{array} & O_6 = \begin{array}{cccc} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0, \end{array} & O_7 = \begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0. \end{array} \end{array}$$

In order for f to be both *RotS* and *CI(1)*, we must first be able to partition \mathbb{V}_n in such a way that

1. Each subset of the partition forms an orthogonal array, and
2. The partition must be rotation symmetric.

The first task is accomplished using the previously outlined partitioning technique which employs a linear orthogonal array along with its cosets, each of which are orthogonal arrays in their own right. However, in order to satisfy the second requirement, we select as our starting point a cyclic orthogonal array. Moreover, once the cosets have been formed, we group them in such a way as to ensure that each group of orthogonal arrays contains all vectors, $\mathbf{x} \in \mathbb{V}_4$ from the same cyclic class, $\langle \mathbf{x} \rangle$. Having done so, we then map all vectors

within each group to the same output value, \mathbb{Z}_q . Given that

$$\begin{aligned}\langle 0001 \rangle &= \{0001, 0010, 0100, 1000\} & \langle 0011 \rangle &= \{0011, 0110, 1100, 1001\} \\ \langle 1110 \rangle &= \{1110, 1101, 1011, 0111\} & \langle 0101 \rangle &= \{0101, 1010\},\end{aligned}$$

we can for example achieve our goal using the following mapping:

$$\{O_0 \rightarrow 0, \{O_1, O_2, O_3, O_4\} \rightarrow 1, O_6 \rightarrow 3, \{O_5, O_7\} \rightarrow 2\}.$$

Doing so produces the *RotS* and *CI(1)* generalized Boolean function in Table 3.10

Table 3.10: A *RotS* and *CI(1)* generalized Boolean function $f \in \mathcal{GB}_4^4$

\mathbb{V}_4	f
0000	0
0001	1
0010	1
0011	2
0100	1
0101	3
0110	2
0111	1
1000	1
1001	2
1010	3
1011	1
1100	2
1101	1
1110	1
1111	0

Algorithm 5 *RotS* and *CI*(*t*) generalized Boolean function construction

```

1: Select a cyclic linear orthogonal array  $O_0 = \overleftarrow{OA}(m, n, 2, t)$ , where  $m = 2^\ell$  and  $n > \ell$ .
2: Store row vectors of  $O_0$  to array  $V$ .
3: Create an array of arrays,  $P$ .
4: Create two arrays  $F$  and  $I$ .
5: Store 1 in  $I$ .
6: for  $i = 1$  to  $2^n - 1$  do
7:    $\mathbf{x} = i_2$ 
8:   if  $\mathbf{x} \notin V$  and  $\mathbf{x} \notin P$  then
9:     Construct set of cyclic vectors  $\langle \mathbf{x} \rangle$ .
10:    Compute  $p_{\mathbf{x}} = |\langle \mathbf{x} \rangle|$ .
11:    Store  $(P_{\mathbf{x}}, \langle \mathbf{x} \rangle)$  to  $P$ .
12:   end if
13:    $i++$ 
14: end for
15: Sort  $P$  such that  $(P_{\mathbf{x}}, \langle \mathbf{x} \rangle)$  tuples appear in ascending order with respect to  $P_{\mathbf{x}}$ .
16: for  $j = 0$  to  $\text{length}.P(\text{outer}) - 1$  do
17:    $\text{cnt} = 0$ 
18:   for  $k = 1$  to  $P[j][0] - 1$  do
19:     if  $P[j][k] \notin V$  then
20:       for  $h = 0$  to  $m - 1$  do
21:          $v_h = V[h] \oplus P[j][k]$ 
22:         Store  $v_h$  to  $V$ 
23:       end for
24:        $\text{cnt}++$ 
25:     end if
26:      $k++$ 
27:   end for
28:   store  $\text{cnt}$  to  $I$ .
29:    $j++$ 
30: end for
31: Set  $q \leftarrow \text{length}.I$ 
32: Create set  $\mathbb{Z}_q = \{0, 1, \dots, q-1\}$ 
33:  $\text{start} \leftarrow 0$ 
34:  $\text{end} \leftarrow m - 1$ 
35: for  $i = 0$  to  $q - 1$  do
36:   Select an output value  $c_i \in \mathbb{Z}_q$ .
37:   for  $k = \text{start}$  to  $\text{end}$  do
38:     Store  $(V[k], c_i)$  to  $F$ .
39:   end for
40:    $\text{start} \leftarrow \text{end} + 1$ 
41:    $\text{end} \leftarrow \text{end} + I[i] \cdot m$ 
42: end for
43: Sort tuples of  $F$  such that input vectors appear in lexicographic order.

```

Proof of Correctness of $RotS$ and $CI(t)$ Generalized Boolean Function Construction:

Suppose we wish to construct a $RotS$ and $CI(t)$ generalized Boolean function $f \in \mathcal{GB}_n^q$ using Algorithm 5. As was the case with Algorithm 4, we begin by selecting a suitable linear orthogonal array. In this case, however, we stipulate that the orthogonal array $O_0 = \overleftarrow{OA}(2^\ell, n, 2, t)$, ($n > \ell$), must also be cyclic. To ensure the function is correlation immune, Algorithm 5 retains the general approach of partitioning \mathbb{V}_n using O_0 along with its cosets, all of which also are orthogonal arrays. However, in order to also achieve rotation symmetry, the way in which we go about creating and grouping these cosets has been slightly modified. For each vector $\mathbf{x} \in \mathbb{V}_n \setminus O_0$, we construct the set of vectors, $\langle \mathbf{x} \rangle$. For each unique cyclic class, $\langle \mathbf{x} \rangle$, we compute its associated period, $p_{\mathbf{x}} = |\langle \mathbf{x} \rangle|$ and store $(p_{\mathbf{x}}, \langle \mathbf{x} \rangle)$ to an array of arrays, P . Once this task has been accomplished, we sort the tuples of P such that the $p_{\mathbf{x}}$ values appear in increasing order. Using vectors in P , Algorithm 5 then forms the $2^{n-\ell} - 1$ cosets of O_0 in the familiar manner. O_0 is a simple linear orthogonal array and its row vectors form a subgroup of \mathbb{V}_n . Using O_0 along with its cosets, the algorithm therefore creates a partition of \mathbb{V}_n . Each coset within the partition is unique and in accordance with Lemma 3.10, also is an $OA(m, n, 2, t)$ orthogonal array. There is however, no guarantee that the cosets are cyclic. Consequently, in order to ensure that cosets which contain vectors belonging to the same cyclic class get grouped together, the algorithm successively builds cosets using the vectors within the same cyclic classes in P , and keeps track of the membership boundaries of the vectors within groupings of cosets using the index array I . To demonstrate that this method of grouping orthogonal arrays produces a rotation symmetric partition of \mathbb{V}_n , we argue as follows: O_0 is a cyclic orthogonal array, hence for every row vector $\mathbf{y} \in O_0$, O_0 contains the set $\langle \mathbf{y} \rangle$ of every vector which is a cyclic rotation of \mathbf{y} . Select a vector \mathbf{x} such that $\mathbf{x} \in \mathbb{V}_n \setminus O_0$, and form the cyclic set $\langle \mathbf{x} \rangle$, containing all possible vectors which are cyclic rotations of \mathbf{x} . Let $\mathbf{z} = \mathbf{y} \oplus \mathbf{x}$. Suppose that for some κ , where $1 \leq \kappa \leq n$, there exist a cyclic rotation ρ_n^κ such that $\rho_n^\kappa(\mathbf{z}) \notin B$, where B is the set defined as $B = \{\mathbf{y} \oplus \mathbf{x} \mid \mathbf{y} \in \langle \mathbf{y} \rangle, \mathbf{x} \in \langle \mathbf{x} \rangle\}$. $\rho_n^\kappa(\mathbf{z}) = \rho_n^\kappa(\mathbf{y} \oplus \mathbf{x}) = \rho_n^\kappa(\mathbf{y}) \oplus \rho_n^\kappa(\mathbf{x})$. Therefore, in order for $\rho_n^\kappa(\mathbf{z}) \notin B$ it would imply that either $\rho_n^\kappa(\mathbf{y}) \notin \langle \mathbf{y} \rangle$ or $\rho_n^\kappa(\mathbf{x}) \notin \langle \mathbf{x} \rangle$, neither of which by definition are possible. We therefore conclude that the set of vectors B is cyclic. Given the fact that O_0 is a subgroup of \mathbb{V}_n , it clearly must contain a minimum of two cyclic classes, namely $\langle \mathbf{0} \rangle$, as well as at least one additional class $\langle \mathbf{y} \rangle$, where $\mathbf{y} \in O_0$. However, the way in which we construct the cosets guarantees that the vectors from all cyclic classes in O_0

are added to all the vectors in the cyclic class $\langle \mathbf{x} \rangle$. Moreover, since O_0 contains the identity element $\mathbf{0}$ we can be assured that each vector within a given cyclic class $\langle \mathbf{x} \rangle$ will appear in a coset of O_0 .

Remark 3.49. In order to avoid constructing duplicate orthogonal arrays, the algorithm takes care to check after each iteration whether or not the next vector in the generating set $\langle \mathbf{x} \rangle$ occurred in the previously constructed coset. For example, If $O_0 = \{0000, 1111\}$ and we were using the set $\langle 0101 \rangle = \{0101, 1010\}$ to form a set of cyclic cosets of O_0 , the first coset constructed (using 0101) would be $\{0101, 1010\}$. However, the second vector $1010 \in \langle 0101 \rangle$ already appeared in the coset produced, therefore the algorithm would not use it again, but rather skip it, determine that the set $\langle 0101 \rangle$ had been exhausted, and write the index of the last vector in the set of cosets to the index array, I , before proceeding to the next array element in P .

Algorithm 5 terminates once the number of vectors in the set V is 2^n . At this point it will contain $2^{n-\ell}$ orthogonal arrays and be a partition of \mathbb{V}_n . The index array I keeps track of how many cosets each cyclic class $\langle \mathbf{x} \rangle$ produces, thus enabling the required grouping of orthogonal arrays. By counting the number of elements in I , the algorithm determines the number of distinct functional output values, q , achievable in the construction. By subsequently assigning the same output value, $c_i \in \mathbb{Z}_q$, for $i = 0$ to $q - 1$, to every vector within a set of orthogonal arrays, the algorithm not only ensures the function is correlation immune (order t), but that it also is rotation symmetric.

Example 3.50. Suppose we wish to construct a *RotS* and *CI*(2) generalized Boolean function $f \in \mathcal{GB}_7^4$. We first select the cyclic $\overleftarrow{OA}(8, 7, 2, 2)$ linear orthogonal array:

$$O_0 = \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}$$

The Algorithm begins by storing the row vectors of O_0 to the array V . It initializes an array of arrays P and initializing an array I with the value 1. For each vectors $\mathbf{x} \in \mathbb{V}_7 \setminus O_0$, the

algorithm checks that $\mathbf{x} \notin P$, then constructs the cyclic set of vectors $\langle \mathbf{x} \rangle$. It subsequently computes the period $P_{\mathbf{x}} = |\langle \mathbf{x} \rangle|$ and stores $(P_{\mathbf{x}}, \{\langle \mathbf{x} \rangle\})$ to P . Once all cyclic sets of vectors that are not in O_0 have been constructed and stored in P along with their associated periods, the algorithm will begin to use the vectors within these cyclic classes to construct the cosets of O_0 . Since in this example n is prime, all vectors in \mathbb{V}_7 are either period 1 or period 7. For any orthogonal array, there are only two period 1 vectors, namely $\mathbf{0}$ and $\mathbf{1}$. The $\mathbf{0}$ vector is the additive identity in $G = (\mathbb{V}_7, \oplus)$, and thus must be in O_0 given the fact that O_0 is a linear orthogonal array and $O_0 < G$. However, as luck would have it, $\mathbf{1}$ is not in O_0 . This means that the first entry in P will be $(1, \{11111111111111\})$ and the first set of cyclic cosets which Algorithm 5 constructs will only include the following cyclic orthogonal array:

$$O_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Once the last of these vectors has been added to the set V , the set of generating vectors within this entry of P have been exhausted. The algorithm will then store the number of cosets which were created, in this case 1, to the index array I , before moving on to the next entry in P , which is:

$$(7, \{0000001, 1000000, 0100000, 0010000, 0001000, 0000100, 0000010\}).$$

Using these vectors, the algorithm in turn constructs and stores the following seven cosets to V :

$$O_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad O_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \quad O_4 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \quad O_5 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{array}{l}
O_6 = \begin{array}{ccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array} \\
O_7 = \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{array} \\
O_8 = \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}
\end{array}$$

Once this is done, the algorithm will save the value 7 to I and then move to the next entry in P , which happens to be:

$$(7, \{0000011, 1000001, 1100000, 0110000, 0011000, 0001100, 0000110\}).$$

Using this set of vectors, the algorithm produces the final seven cosets:

$$\begin{array}{l}
O_9 = \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{array} \\
O_{10} = \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{array} \\
O_{11} = \begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \\
O_{12} = \begin{array}{ccccccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array} \\
O_{13} = \begin{array}{ccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \\
O_{14} = \begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array} \\
O_{15} = \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}
\end{array}$$

Once these cosets have been saved to V , the algorithm stores the value 7 to I . Having stored all 2^n vectors to V , the loop that builds cosets terminates. Using the array I , the algorithm then determines the number of sets, q , into which the orthogonal arrays were grouped. For each of these groups, it chooses a value from $c_i \in \mathbb{Z}_q$, $i = 0$ to $q - 1$. Using I it computes the start and end boundaries for each group of vectors and for $k = start$ to end within each group it saves $(V[k], c_i)$ to a function array F . Due to the considerable size of the function, we omit, in the interest of space, a complete table of input and output values and represent instead the mapping created by the algorithm:

$$\{O_0 \rightarrow c_0, O_1 \rightarrow c_1, \{O_2, \dots, O_8\} \rightarrow c_2, \{O_9, \dots, O_{15}\} \rightarrow c_3\},$$

that guarantees that our function is both *RotS* and *CI*(2).

Lemma 3.51. *Given a cyclic linear orthogonal array $O = \overleftarrow{OA}(2^{n-1}, n, 2, t)$, the remaining set of vectors $\mathbb{V}_n \setminus O$ also forms a cyclic orthogonal array, $O = \overleftarrow{OA}(2^{n-1}, n, 2, t)$.*

Proof. The proof uses an argument similar to the one found in Lemma 3.25. Let O_0 be a cyclic linear orthogonal array $\overleftarrow{OA}(2^{n-1}, n, 2, t)$. Since O_0 is a linear orthogonal array, the row vectors of O_0 form an order 2^{n-1} abelian subgroup of \mathbb{V}_n under \oplus . Select a vector $\mathbf{a} \in \mathbb{V}_n$ not present in O_0 and add it in turn to each row vector in O_0 thereby forming the coset, O_1 , to O_0 . Then $O_0 \cup O_1 = \mathbb{V}_n$ and according to Lemma 3.10, O_1 is also a $OA(2^{n-1}, n, 2, t)$ orthogonal array. Since O_0 is cyclic, for all row vectors $\mathbf{x} \in O_0$, $\langle \mathbf{x} \rangle \subseteq O_0$. Thus, for all remaining row vectors $\mathbf{y} \in \mathbb{V}_n \setminus O_0$ it must be the case that $\langle \mathbf{y} \rangle \subseteq \mathbb{V}_n \setminus O_0$, proving that O_1 also is a cyclic $\overleftarrow{OA}(2^{n-1}, n, 2, t)$ orthogonal array. ■

Theorem 3.52. *Let $O_0 = \overleftarrow{OA}(2^\ell, p, 2, t)$ be a cyclic linear orthogonal array, where p is prime and $p > \ell + 1$. If $\mathbf{1} \notin O_0$, then it is always possible, using Algorithm 5, to create a *RotS* and *CI*(t) generalized Boolean function, $f \in \mathcal{GB}_p^q$, where q is at least 3.*

Proof. O_0 is a linear orthogonal array, so it along with its cosets will partition \mathbb{V}_p into $2^{p-\ell} \geq 4$ orthogonal arrays of strength t . Since O_0 is cyclic, $\mathbf{1}$ is a period 1 vector, and $\mathbf{1} \notin O_0$, we can form the cyclic coset O_1 using $\mathbf{1}$. Although the remaining $2^{p-\ell} - 2$ cosets may not be cyclic, by assigning distinct output values $c_i \in \mathbb{Z}_3$ for $i = 0$ to 2 such that:

$$\{O_0 \rightarrow c_0, O_1 \rightarrow c_1, \{O_2, \dots, O_{2^{p-\ell}-1}\} \rightarrow c_2\},$$

we produce a *RotS* and *CI*(t) generalized Boolean function $f \in \mathcal{GB}_p^3$. In the event there exist s additional cyclic cosets in the set $\{O_2, \dots, O_{2^{p-\ell}-1}\}$, then we can construct a *RotS* and *CI*(t) generalized Boolean function $f \in \mathcal{GB}_p^q$, where $q \leq 3 + s$. ■

Definition 3.53. We adopt Cusick and Stănică's notion from [11, p. 113] and denote g_n as the cardinality of the partition of \mathbb{V}_n into cyclic classes.

Cusick and Stănică provide the following formulae for g_n in Theorem 5.68 and Corollary 5.69 of [11, p. 127]. We make use of these result in subsequent theorems and thus include their result here, albeit without proof. The interested reader may refer to their cited work as well as [45] and [46] for proofs and further discourse on the stated results.

Theorem 3.54. [11, p. 127]

$$g_n = \frac{1}{n} \sum_{\tau|n} \phi(\tau) 2^{n/\tau},$$

where $\phi(\tau)$ is Euler's phi-function.

If $n = p$, p prime, it possible to obtain a simpler expression. In this case,

$$g_p = \frac{1}{p} \sum_{\tau|p} \phi(\tau) 2^{p/\tau} = 2 + \frac{2^p - 2}{p}.$$

Lemma 3.55. The number of possible RotS generalized Boolean functions in \mathcal{GB}_n^q is at most $g(n)^{g(n)}$.

Proof. In order to construct a RotS generalized Boolean function, we partition \mathbb{V}_n into cyclic classes, of which there are $g(n)$. All vectors within each cyclic class is mapped to the same output in \mathbb{Z}_q . For each partition there are q choices for the output values. Thus, all told there are $q^{g(n)}$ possible functions. Since $q \leq g(n)$. The result is established. ■

Lemma 3.56. If a linear orthogonal array of the form $OA(2, p, 2, 1)$, where p is an odd prime, is used to construct a cyclic partition of \mathbb{V}_p containing 2^{p-1} orthogonal arrays, then the maximum obtainable number of subsets is $1 + \frac{2^{p-1}-1}{p}$.

Proof. Since p is prime, each vector in \mathbb{V}_p is either period 1 or period p , and Theorem 3.54 tells us that there will be a total of $2 + \frac{2^p-2}{p}$ cyclic classes. The construction requires that each orthogonal arrays consists of two vectors $\mathbf{x} \in \mathbb{V}_p$ and its complement $\bar{\mathbf{x}}$. Each cyclic class of vectors $\langle \mathbf{x} \rangle$ is therefore grouped with $\langle \bar{\mathbf{x}} \rangle$, thus causing the total number of subsets in the partition to be:

$$\left(2 + \frac{2^p - 2}{p}\right) / 2 = 1 + \frac{2^{p-1} - 1}{p}.$$

■

Theorem 3.57. *The number of possible RotS and CI(1) generalized Boolean functions, $f \in \mathcal{GB}_p^q$, $q \leq 1 + \frac{2^{p-1}-1}{p}$, constructed using a linear orthogonal array of the form $OA(2, p, 2, 1)$, where p is an odd prime is:*

$$\left(1 + \frac{2^{p-1} - 1}{p}\right)^{1 + \frac{2^{p-1} - 1}{p}}.$$

Proof. Observe that for all p , the number of orthogonal arrays, 2^{p-1} , in the partition is strictly greater than $1 + \frac{2^{p-1}-1}{p}$. By applying Lemmas 3.55 and 3.56 the result immediately follows. ■

Remark 3.58. A surprising consequence of Conjecture 2.26 should it prove to be true, is that balanced and symmetric generalized Boolean functions, where $q > 2$, do not exist. This however, is not the case with balanced and RotS generalized Boolean functions.

Example 3.59. Consider constructing a 4-variable RotS generalized Boolean function. We partition \mathbb{V}_4 into its 6 cyclic classes: $\langle 0000 \rangle$, $\langle 1111 \rangle$, $\langle 0101 \rangle$, $\langle 0001 \rangle$, $\langle 0011 \rangle$, $\langle 0111 \rangle$, of respective periods 1, 1, 2, 4, 4, 4. Therefore, by mapping the classes of input vectors to output values in \mathbb{Z}_4 in the following manner, we create a balanced RotS generalized Boolean function $f \in \mathcal{GB}_4^4$:

$$\{\{\langle 0000 \rangle, \langle 1111 \rangle, \langle 0101 \rangle\} \rightarrow c_0, \langle 0001 \rangle \rightarrow c_1, \langle 0011 \rangle \rightarrow c_2, \langle 0111 \rangle \rightarrow c_3\},$$

where c_i , with $i = 0$ to 3, are distinct values in \mathbb{Z}_4 .

Lemma 3.60. *For an odd prime p and $k > 2$, it is not possible to partition \mathbb{V}_p into k equally sized cyclic subsets.*

Proof. Since p is prime, the only possible periods for vectors in \mathbb{V}_p are 1 or p . The only two period 1 vectors in \mathbb{V}_p are $\mathbf{0}$ and $\mathbf{1}$. All remaining vectors have period p . We wish to partition \mathbb{V}_p into k subsets, each of which is cyclic. All vectors within a given cyclic class must therefore be contained in the same subset. However, since $k > 2$ and p is an odd prime, there is no way in which to distribute $\mathbf{0}$ and $\mathbf{1}$ among the k subsets which will ensure they all are of equal cardinality. ■

Theorem 3.61. *There are no balanced and RotS generalized Boolean functions $f \in \mathcal{GB}_p^q$, for odd prime, p , and $q > 2$.*

Proof. The result is an immediate consequence of Lemma 3.60. ■

CHAPTER 4:

Avalanche Criteria for Generalized Boolean Functions

War is the realm of uncertainty.
Information is the resolution of
uncertainty. Cryptology is the gateway
between these entropy states.

Carl von Clausewitz, Claude Shannon,
and yours truly ✍

4.1 Introduction

It is important that functions that are used in cryptographic applications are resistant to attacks involving the use of knowledge of the input to infer anything about the output. In the preceding chapter we examined correlation immunity properties of generalized Boolean functions. We will now explore the so-called “avalanche effect” whereby a small change in the input of a function results in a large, but in some sense uniform, change to the output of the function. Such a condition, now referred to as the strict avalanche criterion was first defined by Webster and Tavares [50] in their research on designing good Substitution boxes (S-boxes). This area of research is of particular relevance to generalized Boolean functions as well, in part due to their potential use as components in look-up tables and S-boxes of future cryptographic systems.

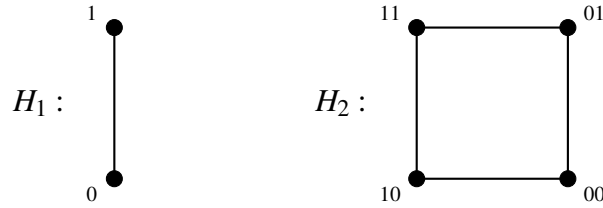
Definition 4.1. [11, p. 31] A Boolean function $f(\mathbf{x})$ in n variables is said to satisfy the *strict avalanche criterion* (SAC) if changing any one of the n bits in the input vector \mathbf{x} changes the output of the function for exactly half of the 2^{n-1} possible input vectors, \mathbf{x} .

4.2 A Strict Avalanche Criterion Construction for Boolean Functions

Given the fact that we will be examining input vectors which differ by a single bit along with their associated functional output values, it is very natural to make use of hypercubes. The idea of enlisting the aid of hypercubes in the study of SAC functions is admittedly not original. It was first adopted by Biss in 1998 [1], albeit with a combinatorial approach and not the graph theoretic point of view which we adopt here.

Definition 4.2. [9, p.25] A hypercube of dimension n , denoted H_n , is the graph whose vertex set is the set of n long binary vectors $\mathbf{x} \in \mathbb{V}_n$ and where two vertices are adjacent in the graph if they differ by exactly one bit.

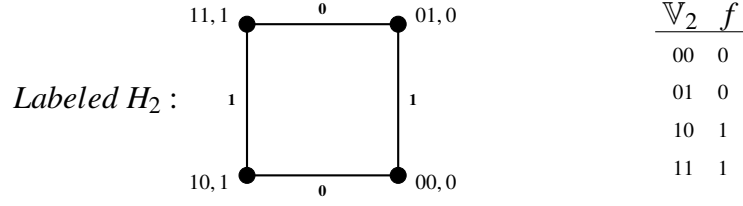
Example 4.3. Below we depict the hypercubes, H_1 and H_2 . Notice that adjacent vertices within each graph differ by one bit:



There is a simple recursive method by which hypercubes can be built. H_2 is obtained by taking two copies of H_1 and connecting the corresponding (similarly labeled) vertices in both graphs. The vertex labels are then updated as follows: In the first copy of H_1 , append 0 to the front of each vector \mathbf{x} , thereby obtaining the new label, "0 \mathbf{x} ". For the second copy of H_1 append 1 to the front of each vector, thus producing the new vector "1 \mathbf{x} ".

We represent a Boolean function $f \in \mathcal{B}_n$ using the n -dimensional hypercube $H_n = (\mathbb{V}_n, E)$, where \mathbb{V}_n is the vertex set and E is the edge set of the graph. Denote $\varepsilon = \{\mathbf{x}_j, \mathbf{x}_h\}$ as an edge in the graph, where $\mathbf{x}_j, \mathbf{x}_h \in \mathbb{V}_n$ are distinct vertices in H_n . We label each vertex $\mathbf{x} \in \mathbb{V}_n$ with the tuple $(\mathbf{x}, f(\mathbf{x}))$, where $f(\mathbf{x}) \in \mathbb{F}_2$. For each edge $\varepsilon \in E$, we label ε with the value 1 if $f(\mathbf{x}_j) = f(\mathbf{x}_h)$ and with 0 otherwise.

Example 4.4. Adopting this approach we represent the below Boolean function $f \in \mathcal{B}_2$ using the depicted labeled graph H_2 :

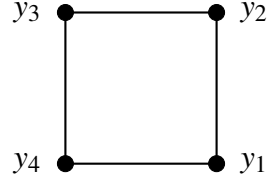


Having established a graph-theoretic frame of reference from which to work, we first consider the conditions under which our labeled hypercube will satisfy the *SAC* feature for Boolean functions. All vertices differing by exactly one bit in H_n are connected by an edge. Moreover, should any pair of such vertices agree with respect to their output values, the edge between them is labeled with a value of 1. Given the fact that the total number of edges in a hypercube is $2^{n-1}n$, it is clear that under this Boolean function model paradigm, a Boolean function will be *SAC* if and only if the sum of the edge set labels of its associated graph H_n equals $2^{n-2}n$. We refer to labeled hypercubes which satisfy this requirement as *SAC* hypercubes.

When attempting to construct *SAC* Boolean functions, one can use the fact that hypercubes can be constructed recursively to one's advantage. By utilizing two appropriately chosen *SAC* hypercubes $H_{n,1}$, $H_{n,2}$, which once connected will have 2^{n-1} newly formed edges labeled with 1's, (in other words, half of $H_{n,1}$ and $H_{n,2}$'s corresponding vertices agree with respect to their output values), the newly formed hypercube H_{n+1} will also be *SAC*. In order to be in a position to carry out such constructions, we must first analyze and derive the *SAC* hypercube "base case" if you will. We do so by contemplating how the vertices of these graphs can be labeled with output values in order to obtain the requisite edge label sum. Considering first H_1 , we see that there clearly is no way in which this can be accomplished, since it only contains one edge. Turning our attention to H_2 , we consider the number of different ways this labeling can be carried out.

Theorem 4.5. *There are 12 possible SAC labelings of the 2 dimensional hypercube.*

Proof. Without loss of generality, we choose to begin labeling at the lower right vertex and proceed counter-clockwise around H_2 . Given the vertex labeling vector $\mathbf{y} = y_1y_2y_3y_4$, where $i = 1$ to 4 and $y_i \in \mathbb{F}_2$, the labeling scheme will thus be as follows:



For $n > 2$, we will use the H_n label vector $\mathbf{w} = \mathbf{y} \parallel \mathbf{z}$, where \mathbf{y} and \mathbf{z} are labeling vectors for hypercubes H_{n-1} .

There are a total of $\sum_{i=0}^4 \binom{4}{i} = 16$ possible vectors \mathbf{y} , which we represent using the following cyclic classes:

$$\begin{aligned} \langle 0000 \rangle &= \{0000\} \\ \langle 0001 \rangle &= \{0001, 0010, 0100, 1000\} \\ \langle 0011 \rangle &= \{0011, 0110, 1100, 1001\} \\ \langle 1110 \rangle &= \{1110, 1101, 1011, 0111\} \\ \langle 0101 \rangle &= \{0101, 1010\} \\ \langle 1111 \rangle &= \{1111\}. \end{aligned}$$

To determine whether a labeling satisfies our requirements, we evaluate \mathbf{y} as follows:

$$\sum_{i=1}^n y_i \cdot y_{i+1},$$

where

$$y_{i+1} = \begin{cases} y_{i+1} & \text{if } i+1 \leq n, \\ y_{i+1-n} & \text{if } i+1 > n. \end{cases}$$

If this sum equals 2, then \mathbf{y} is acceptable, otherwise it is not. Among the possible labelings, 0000 and 1111 will of course not work, and neither will the labelings from the set $\langle 0101 \rangle$. The remaining 12 labelings represented here by their cyclic classes $\langle 0001 \rangle$, $\langle 0011 \rangle$, and $\langle 1110 \rangle$ all satisfy the requirement we seek. Hence, any one of them when applied to H_2 will produce a SAC hypercube of dimension 2, and thus also represent a SAC 2-variable Boolean function. ■

Remark 4.6. Using the labeling $\mathbf{y} = 0011$ produces the SAC hypercube H_2 and associated

Boolean function depicted in Example 4.4.

As previously suggested, we can use two appropriately chosen 2-dimensional *SAC* hypercubes to construct a 3-dimensional *SAC* hypercube. In order to ease the selection task we demonstrate a quick verification procedure which takes advantage of our consistent labeling scheme. Let \mathbf{y}_1 and \mathbf{y}_2 be two of the 12 *SAC* labeling schemes for H_2 . In order to determine whether or not they, once connected, will produce a 3-dimensional *SAC* hypercube, we evaluate the two vectors using the following label comparator function, $f(\mathbf{y}_1, \mathbf{y}_2) = wt(\mathbf{y}_1 \oplus \mathbf{y}_2)$. The function compares label values at corresponding indices using *XOR*. Hence similar values fail to contribute anything to the Hamming weight of the resultant vector whereas dissimilar label values add 1. Consequently, if in our case $f(\mathbf{y}_1, \mathbf{y}_2) = 2$ (namely half of the vertices), then given the fact that each of the original H_2 hypercubes were at the onset *SAC*, one can be assured that the sum of the edge-set labels for the resultant 3-dimensional hypercube H_3 will achieve the requisite $2^{n-2}n$ value and thus satisfy the strict avalanche criterion.

Theorem 4.7. *There are a total of 56 labeled SAC 3-dimensional hypercubes with SAC labeled H_2 subgraphs.*

Proof. According to Theorem 4.5, there are 12 2-dimensional *SAC* labeled hypercubes. Each of these has two edges labeled with 1's (and 2 with 0's). Moreover, we know that in order for the labeled H_3 hypercube to be *SAC*, 6 of its 12 edges must also be labeled with 1's. Therefore when connecting the two H_2 hypercubes, we must ensure that 2 of their 4 corresponding vertices agree with respect to their output labels. Using the previously described comparator function, $f(\mathbf{y}_1, \mathbf{y}_2) = wt(\mathbf{y}_1 \oplus \mathbf{y}_2)$, we could of course exhaustively evaluate the relatively small set of label vectors to obtain the stated result. However, we choose instead to arrive at the answer using a counting argument. We evaluate in turn each of the three cyclic classes, $\langle 0001 \rangle$, $\langle 0011 \rangle$ and $\langle 1110 \rangle$. Beginning with $\langle 0001 \rangle$ we consider the possible vector pairings which, when added modulo 2, will produce a vector of weight 2. Let $\mathbf{y} = 0001$. Since \mathbf{y} is of Hamming weight 1, $wt(\mathbf{y} \oplus \rho^\kappa(\mathbf{y})) = 2$ for $\kappa = 1$ to 3. There are 4 vectors in $\langle 0001 \rangle$ for which this works, so there are $4 \cdot 3 = 12$ such possible pairings. Adding a Hamming-weight-2 vector to a Hamming-weight-1 vector always produces a vector of Hamming weight 1 or 3, so we may readily disregard this

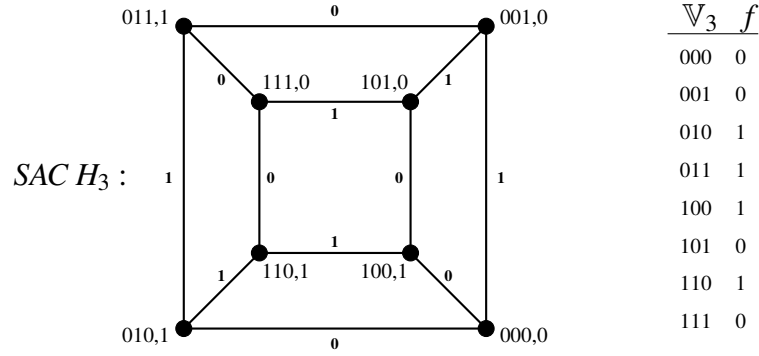
possibility. Let $\mathbf{z} = 1110$. Then $wt(\mathbf{y} \oplus \rho^\kappa(\mathbf{z})) = 2$, for $\kappa = 1$ to 3. As before, there are 4 vectors in $\langle 0001 \rangle$ for which this works, so there $4 \cdot 3 = 12$ such possible pairings. Observe that $\bar{\mathbf{y}} = 1110$, so the analysis is identical for this cyclic class. Finally we consider $\langle 0011 \rangle$. Adding two Hamming weight 2 vectors together produces either a Hamming weight 0, 2 or 4 vector. The first and last stated possibilities can each only happen once, so among the 4 possible shifts of 0011, it must be the case that the middle condition occurs twice. Thus there are $4 \cdot 2 = 8$ such possible pairings. Having exhausted all possibilities within the 3 cyclic classes, we tally the possible pairings which yields $2(12 + 12) + 8 = 56$. ■

Remark 4.8. The discourse above highlights a useful *SAC* H_3 construction strategy. Select a vector, \mathbf{y} , from any of the three cyclic classes $\langle 0001 \rangle$, $\langle 0011 \rangle$ or $\langle 1110 \rangle$. If $wt(\mathbf{y}) = 1$ or $wt(\mathbf{y}) = 3$, then \mathbf{y} along with a cyclic shift, $\rho^\kappa(\mathbf{y})$, for $\kappa = 1$ to 3, will always ensure $wt(\mathbf{y} \oplus \rho^\kappa(\mathbf{y})) = 2$. If $wt(\mathbf{y}) = 2$, then any odd shift ($\kappa = 1$ or $\kappa = 3$) will result in $wt(\mathbf{y} \oplus \rho^\kappa(\mathbf{y})) = 2$.

Example 4.9. Suppose we wish to construct a Boolean function $f \in \mathcal{B}_3$ which satisfies the strict avalanche criterion. We begin by first selecting two H_2 labelings $\mathbf{y} = 0011$ and $\mathbf{z} = 1001$. Before proceeding, we confirm that $\sum_{i=0}^n y_i \cdot y_{i+1} = 2$ and $\sum_{i=0}^n z_i \cdot z_{i+1} = 2$. Once complete, we then verify that, once connected, the two \mathbf{y} and \mathbf{z} labeled H_2 hypercubes will produce a *SAC* H_3 labeled hypercube. Given the fact that

$$f(\mathbf{y}, \mathbf{z}) = wt(0011 \oplus 1001) = wt(1010) = 2,$$

we can be assured that this will indeed be the case. We thus proceed to construct the 3-dimensional hypercube H_3 in the standard manner. Doing so, the vertex labels for each H_2 component are augmented with a 0 or 1 in the previously described manner, however, the associated vertex output values for each copy of H_2 remains unchanged. Doing so produces the following graph and associated function truth table:



Having demonstrated the construction technique, we codify this SAC Boolean function construction in the following algorithm:

Algorithm 6 SAC Boolean function construction

```

1: Given two SAC  $H_{n-1}$  binary output labeling vectors  $\mathbf{y}$  and  $\mathbf{z}$ , store them as arrays  $Y$  and  $Z$ .
2:  $m \leftarrow n - 1$ 
3:  $YLength \leftarrow 2^m$ 
4:  $Edge \leftarrow 0$ 
5: Initialize two arrays  $W$  and  $F$  of length  $2YLength$ .
6: for  $i = 0$  to  $YLength - 1$  do
7:   if  $Y[i] == Z[i]$  then
8:      $Edge++$ 
9:   end if
10: end for
11: if  $Edge = 2^{m-1}$  then
12:   for  $i = 1$  to  $YLength$  do
13:     if  $i \equiv 3 \pmod{4}$  then
14:        $W[i-1] \leftarrow Y[i]$ 
15:        $W[i] \leftarrow Y[i-1]$ 
16:        $W[YLength+i-1] \leftarrow Z[i]$ 
17:        $W[YLength+i] \leftarrow Z[i-1]$ 
18:     else
19:        $W[i-1] \leftarrow Y[i-1]$ 
20:        $W[YLength+i-1] \leftarrow Z[i-1]$ 
21:     end if
22:   end for
23: else
24:   Return: "Error! The vectors will not produce a SAC function."
25: end if
26: for  $j = 0$  to  $2YLength - 1$  do
27:    $F[j] \leftarrow (j_2, W[j])$ 
28: end for

```

Proof of Correctness of the SAC Boolean Function Construction:

The algorithm accepts two binary output labeling vectors \mathbf{x} and \mathbf{y} for two hypercubes of dimension $m = n - 1$, storing them in the arrays Y and Z . Since each of these labelings produces a SAC labeled hypercube, we know that each of these hypercubes must contain $2^{n-3}(n-1)$ 1-labeled edges. There will be a total of 2^{n-1} new edges formed once the two hypercubes are connected. Therefore, if half of the corresponding labeled vertices in each hypercube agree with respect to their output values (labels), then 2^{n-2} new edges will be labeled with 1's. The total number of 1-labeled edges in the resultant n -dimension hypercube will therefore be

$$2 \cdot 2^{n-3}(n-1) + 2^{n-2} = 2^{n-2}(n-1+1) = 2^{n-2}n = \frac{2^{n-1}n}{2}.$$

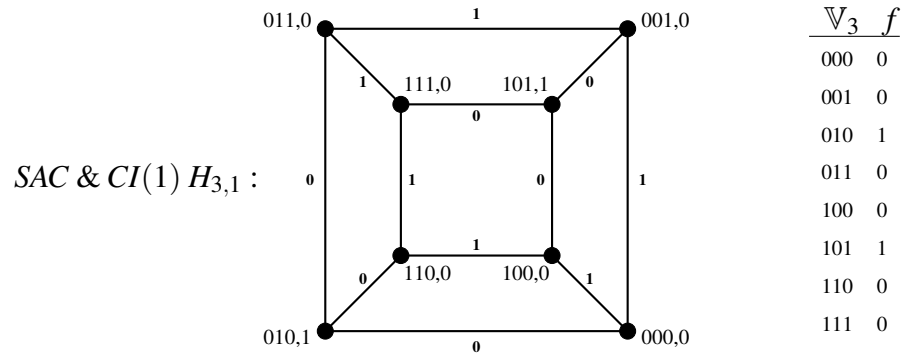
This is exactly half of the total number of edges of the newly-formed hypercube. We therefore conclude that it, along with its corresponding function $f \in \mathcal{B}_n$, must be SAC. Thus, the task at hand is to ensure that exactly half of the corresponding vertex labels in Y and Z agree. This check is carried out in steps 6 to 10 of the algorithm. For $i = 0$ to $YLength - 1$ the algorithm compares array elements $Y[i]$ and $Z[i]$ and increments the *Edge* counter if the values match. If $Edge = 2^{n-1}$, the construction will succeed and the algorithm proceeds to build the function truth table. The adopted labeling schemes, discussed in Theorem 4.5, stores the vector labels in Y and Z as counterclockwise 4-cycles of H_2 planes, so before doing so, it is necessary to store the output values (labels) in lexicographic order in an array W . This procedure is accomplished in steps 12 to 22 of the algorithm. Finally, using W , and for $j = 0$ to $2^n - 1$, the algorithm populates the truth table array F with tuples, $(j_2, W[j])$, of binary input vectors and q -ary output values.

Remark 4.10. The similarity between the Siegenthaler correlation immunity construction outlined in Theorem 3.35 and the SAC hypercube construction from Algorithm 6 should not be lost on the reader. The SAC construction not only uses two graphs (functions) of dimension $n - 1$ to create a graph (function) of dimension n , but like Siegenthaler's it also requires that the frequency of the two output values 0 and 1 agree between the dimension $n - 1$ subgraphs.

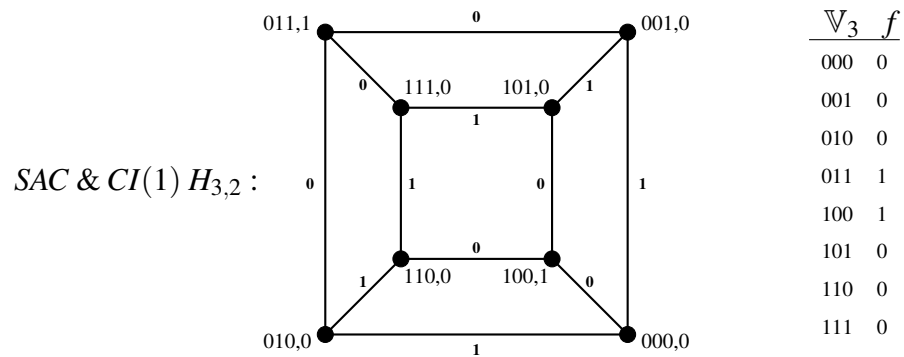
Example 4.11. Suppose we wish to construct a SAC and $CI(1)$ Boolean function $f \in \mathcal{B}_3$. We begin by selecting two H_2 labeling vectors $\mathbf{y} = 0001$ and $\mathbf{z} = 0100$.

Nota bene: The reader is cautioned that, unlike in the case of lexicographic ordering, our labeling scheme reverses the order of 10 and 11. Therefore, although by inspection $\mathbf{y} \parallel \mathbf{z}$ does not (based on symmetry) immediately appear to be $CI(1)$, it in fact is.

As in the previous example, we confirm that $\sum_{i=0}^n y_i \cdot y_{i+1}$ and $\sum_{i=0}^n z_i \cdot z_{i+1}$ both equal 2 and that $f(\mathbf{y}, \mathbf{z}) = wt(0001 \oplus 0100) = wt(0101) = 2$. Having done so, we then construct the H_3 labeled graph below.



Since our construction used \mathbf{y} and \mathbf{z} , which when taken in concert was $CI(1)$, we were not only able to construct a SAC labeled hypercube, but it also turned out to be (order 1) correlation immune. Had we instead chosen the vectors $\mathbf{u} = 0010$ and $\mathbf{v} = 1000$, we would have produced the following SAC and $CI(1)$ hypercube.



Having these two SAC and $CI(1)$ labeled H_3 graphs at our disposal, we demonstrate how to go about combining the Siegenthaler construction of Theorem 3.35 and Algorithm 6 to

produce a Boolean function in 4 variables which is both *SAC* and *CI*(1).

Let f_1 and f_2 be the Boolean functions corresponding to $H_{3,1}$ and $H_{3,2}$. Let $n = 3$ and $\mathbf{w} = \mathbf{y} \parallel \mathbf{z}$ and $\mathbf{t} = \mathbf{u} \parallel \mathbf{v}$. Before merging the two graphs and creating the function in $n + 1$ variables, $f = f_1 \parallel f_2$, we ensure the following hold:

1. $H_{3,1}$ and $H_{3,2}$ are both of proper dimension and *SAC*.
2. $wt(\mathbf{w} \oplus \mathbf{t}) = 2^{n-1}$.
3. For the set of input vectors $\mathbf{x} \in \mathbb{V}_3$, $Pr(f_1(\mathbf{x}) = 0) = Pr(f_2(\mathbf{x}) = 0)$.
4. f_1 and f_2 are both *CI*(1).

With all of these requirements met, we proceed with the construction and create the following labeled hypercube H_4 along with its associated Boolean function truth table:

SAC & *CI*(1) H_4 :

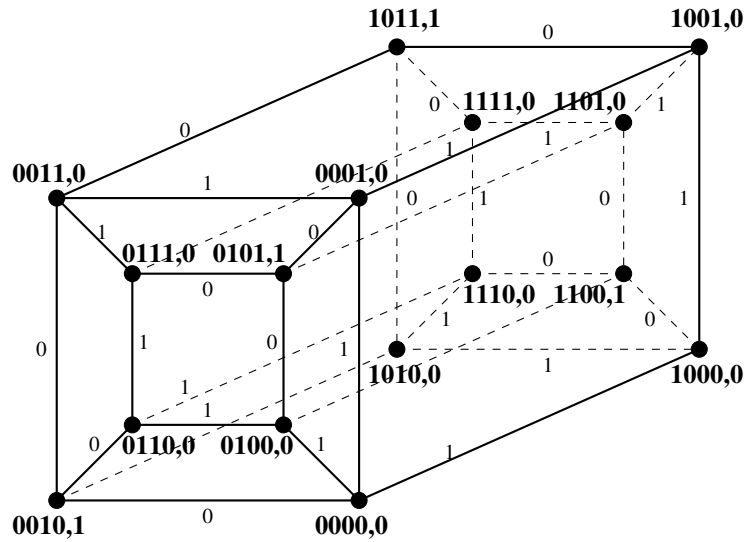


Table 4.1: A *SAC* and *CI(1)* Boolean function $f \in \mathcal{B}_4$

\mathbb{V}_4	f
0000	0
0001	0
0010	1
0011	0
0100	0
0101	1
0110	0
0111	0
1000	0
1001	0
1010	0
1011	1
1100	1
1101	0
1110	0
1111	0

4.3 A Probabilistic Strict Avalanche Criterion

Motivated by the work of Kam and Davika on permutation-substitution networks [22], as well as that of Feistel [14], Webster and Tavares first investigated the strict avalanche criterion in 1986 in an effort to design S-boxes with desirable cryptographic properties. Given the fact that Boolean functions are often employed as components in S-box design, there has subsequently been a great deal of research carried out on *SAC* Boolean functions. In this section, we will seek to extend the notion of the strict avalanche criterion to that of generalized Boolean functions. Throughout the discourse we continue to build upon the graph-theoretic framework previously developed for the Boolean case.

The strict avalanche criterion requires, in the Boolean case, that each output bit should change with probability $1/2$ whenever a single bit of a binary input vectors is comple-

mented [50]. In the generalized Boolean case, we modify the criterion as follows:

Definition 4.12. A generalized Boolean function $f \in \mathcal{GB}_n^q$ is said to satisfy the *probabilistic strict avalanche criterion (PSAC)*, if changing any one of the n bits in an input vector $\mathbf{x} \in \mathbb{V}_n$ results in the output of the function remaining invariant with probability $1/q$.

Remark 4.13. As previously demonstrated, for each Boolean function, it is possible to construct a corresponding labeled hypercube H_n . Consequently, given Definition 4.12, a generalized Boolean function $f \in \mathcal{GB}_n^q$ can only be *PSAC* if $q|2^{n-1}n$. In other words, the number of edges in the graph H_n must be divisible by q .

Example 4.14. We motivate this probabilistic notion of *SAC* using the following example. Suppose we wish to construct a *PSAC* generalized Boolean function $f \in \mathcal{GB}_3^3$. The first task is to verify that the number of edges in H_3 , namely $2^{3-1}3 = 12$, is divisible by 3. This being the case, we proceed. As with the previous *SAC* Boolean function construction, we base our construction on two, albeit not necessarily *PSAC* hypercubes, of dimension $n - 1$. The function's output values are now in \mathbb{Z}_3 . Suppose the two ternary label vectors are $\mathbf{y} = 0011$ and $\mathbf{z} = 2200$. In the case of binary vectors, we had a straightforward method of checking the suitability of a given label vector using the sum of the binary product of its bits. In the generalized Boolean function case, we utilize the same basic idea. However, due to the q -ary nature of the task at hand, we employ the Kronecker delta function instead. Thus, given a vector $\mathbf{y} = (y_1, y_2, \dots, y_n)$, let

$$\delta(y_i, y_{i+1}) = \begin{cases} 0 & \text{if } y_i \neq y_{i+1}, \\ 1 & \text{if } y_i = y_{i+1} \end{cases}$$

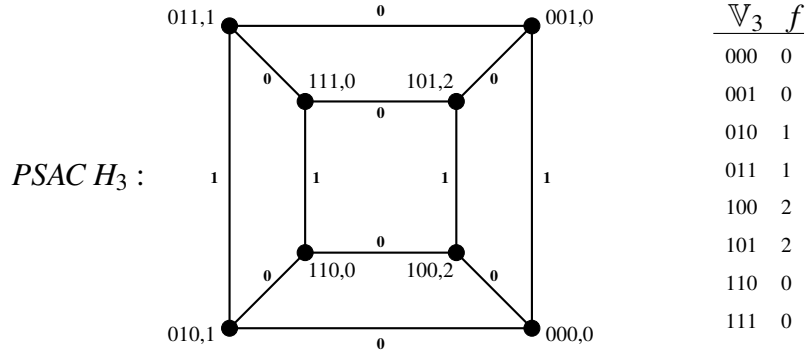
and

$$y_{i+1} = \begin{cases} y_{i+1} & \text{if } i+1 \leq n, \\ y_{i+1-n} & \text{if } i+1 > n. \end{cases}$$

Having previously been given the label vectors \mathbf{y} and \mathbf{z} , we are now capable of computing the number of 1-labeled edges in each of the respective H_2 graphs, id est

$$\sum_{i=0}^n \delta(y_i, y_{i+1}) = 2 \quad \text{and} \quad \sum_{i=0}^n \delta(z_i, z_{i+1}) = 2.$$

We subsequently need to check the number of newly formed edges which will be 1-labeled when the two H_2 graphs are connected. Once again, we need to revise the way in which this is accomplished. Rather than using the XOR operation and computing $wt(\mathbf{y} \oplus \mathbf{z})$, as we did when dealing with Boolean functions, we again avail ourselves of the Kronecker delta and compute instead the sum, $\sum_{i=1}^n \delta(y_i, z_i)$. Doing so, we discover that connecting the two H_2 graphs will not produce any additional 1-labeled edges. Thus, the total number of 1-labeled edges in H_3 will be 4. This in turn means that the probability of an edge being 1-labeled, and thus neighbor vertices within H_3 having the same output label, is $4/12 = 1/3$. The H_2 labeled subgraphs will therefore produce the desired result. We display the following *PSAC* labeled hypercube H_3 along with its associated function truth table.



Having demonstrated our approach to constructing *PSAC* generalized Boolean functions, we now codify the procedure in Algorithm 7.

Remark 4.15. Despite being rather long, Algorithm 7 is, at its core, relatively straightforward. The general approach mirrors that of Example 4.13 and involves using the supplied label vectors to count the number of 1-labeled edges within each subgraph (hypercube of dimension $n - 1$) along with the number of 1-labeled edges which emerge once the two subgraphs are connected. If this number ends up equaling $(2^{n-1}n)/q$, q being the number of different output values (labels) of the generalized function $f \in \mathcal{GB}_n^q$, then we know that according to Definition 4.12, f will satisfy the probabilistic strict avalanche criterion.

Algorithm 7 PSAC generalized Boolean function construction

```
1:  $m \leftarrow n - 1$ 
2: if  $2^m(m+1) \not\equiv 0 \pmod{q}$  then
3:   Print: "Error! Function parameters not capable of producing a PSAC function."
4: else
5:   Store two  $H_{n-1}$  labeling vectors,  $\mathbf{y}$  and  $\mathbf{z}$  as arrays  $Y$  and  $Z$ .
6:    $YLength \leftarrow 2^m$ 
7:   Initialize arrays  $W$  and  $F$  of length  $2YLength$ .
8:    $4Sections \leftarrow YLength/4$ 
9:    $YEdge \leftarrow 0$ 
10:   $ZEdge \leftarrow 0$ 
11:   $TargetCnt \leftarrow (2^{m+1}(m+1))/q$ 
12:  for  $k = 0$  to  $4Sections - 1$  do
13:    for  $j = 4k$  to  $4(k+1) - 1$  do
14:       $EndIndex = j + 1$ 
15:      if  $EndIndex \geq 4$  then
16:         $EndIndex = EndIndex - 4$ 
17:      end if
18:      if  $Y[j] == Y[EndIndex]$  then
19:         $YEdge++$ 
20:      end if
21:      if  $Z[j] == Z[EndIndex]$  then
22:         $ZEdge++$ 
23:      end if
24:    end for
25:  end for
26:  for  $h = 0$  to  $m - 1$  do
27:     $Stepsize \leftarrow 2^{h+2}$ 
28:     $End \leftarrow \frac{YLength}{2^{Stepsize}} - 1$ 
29:    for  $k = 0$  to  $End$  do
30:      for  $j = (2k)Stepsize$  to  $(2k+1)Stepsize - 1$  do
31:        if  $Y[j] == Y[j + Stepsize]$  then
32:           $YEdge++$ 
33:        end if
34:        if  $Z[j] == Z[j + Stepsize]$  then
35:           $ZEdge++$ 
36:        end if
37:      end for
38:    end for
39:  end for
40:   $EdgeCnt = YEdge + ZEdge$ 
41:   $ConnectTarget = TargetCnt - EdgeCnt$ 
42:  if  $EdgeCnt > TargetCnt$  or  $ConnectTarget > 2^m$  then
43:    Print: "Error:  $\mathbf{y}$  and  $\mathbf{z}$  cannot produce a PSAC function."
44:  else
45:    for  $i = 0$  to  $2^m - 1$  do
46:      if  $Y[i] == Z[i]$  then
47:         $EdgeCnt++$ 
48:      end if
49:    end for
50:  end if
51:  if  $EdgeCnt != TargetCnt$  then
52:    Print: "Error:  $\mathbf{y}$  and  $\mathbf{z}$  cannot produce a PSAC function."
53:  end if
54:  for  $i = 1$  to  $YLength$  do
55:    if  $i \equiv 3 \pmod{4}$  then
56:       $W[i-1] \leftarrow Y[i]$ 
57:       $W[i] \leftarrow Y[i-1]$ 
58:       $W[YLength+i-1] \leftarrow Z[i]$ 
59:       $W[YLength+i] \leftarrow Z[i-1]$ 
60:    else
61:       $W[i-1] \leftarrow Y[i-1]$ 
62:       $W[YLength+i-1] \leftarrow Z[i-1]$ 
63:    end if
64:  end for
65:  for  $j = 0$  to  $2YLength - 1$  do
66:     $F[j] \leftarrow (j_2, W[j])$ 
67:  end for
68:  Print:  $F$ 
69: end if
```

Proof of Correctness of the PSAC Generalized Boolean Function Construction:

The first thing the algorithm does, in step 2, is to verify that the number of edges, $2^{n-1}n$, of the resultant graph is divisible by the number of desired number output values (labels), q . If this is satisfied, the algorithm then accepts two label vectors \mathbf{y} and \mathbf{z} , each of length 2^{n-1} , for the two H_{n-1} subgraphs and stores them in arrays Y and Z . Following some required initialization, the algorithm uses Y and Z and begins to compute the number of 1-labeled edges within each labeled H_2 subgraph. Our adopted labeling schemes, discussed in Theorem 4.5, stores vectors labels as counterclockwise 4-cycles. Consequently, in order to begin comparing label values and count corresponding 1-labeled edges within each vector, we must first split the vectors into sub-vectors of length 4 and cyclically check for value agreements. This procedure is carried out in steps 12 to 25 of the algorithm. Once this has been completed the algorithm then needs to check output value agreement for corresponding vertices residing in different planes of each H_{n-1} subgraph. This procedure is accomplished in steps 26 to 39. Upon completion of these steps, the algorithm now has 1-labeled edge counts for both Y and Z which are added together and stored as $EdgeCnt$. $EdgeCnt$ is then subtracted from $TargetCnt$ (the number of 1-labeled edges required in order for H_n to be PSAC). This value is stored as $ConnectTarget$. The algorithm then performs two checks: First, it ensures that $EdgeCnt \leq TargetCnt$. Secondly, it verifies that $ConnectTarget < 2^m$, where 2^m is the number of new edges formed once the two $n - 1$ dimension hypercubes are connected. If either of these conditions fail, then H_n cannot be PSAC and no further computation is needed. If, on the other hand, these conditions are satisfied, the algorithm compares the elements of $Y[i]$ and $Z[i]$ and increments $EdgeCnt$ each time an agreement is encountered. Thus once complete, the algorithm will have a complete tally of the number of 1-labeled edges in the H_n hypercube. By comparing $EdgeCnt$ with $TargetCnt$ a final determination can then be made as to whether or not the construction will produce a PSAC hypercube of dimension n . If the two values prove to be equal, steps 54 to 64 of the algorithm then store, in lexicographic order, the output values of Y and Z in the array W . Using W , and for $j = 0$ to $2^n - 1$, the algorithm finally populates the array F with tuples, $(j_2, W[j])$, of binary input vectors and q -ary output values.

Example 4.16. Suppose we wish to construct a generalized Boolean function $f \in \mathcal{GB}_4^4$ which satisfies the probabilistic strict avalanche criterion. The number of edges in H_4 , namely $2^{4-1}4 = 32$, is divisible by the desired number of output values which is 4, so the

PSAC H₄ :



Table 4.2: A PSAC generalized Boolean function $f \in \mathcal{GB}_4^4$

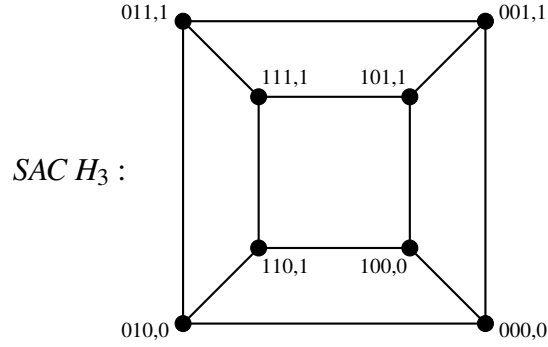
\mathbb{V}_4	f
0000	0
0001	0
0010	2
0011	1
0100	0
0101	2
0110	3
0111	2
1000	2
1001	0
1010	1
1011	0
1100	3
1101	0
1110	2
1111	2

Theorem 4.17. A generalized Boolean function $f \in \mathcal{GB}_n^q$ can only satisfy the probabilistic strict avalanche criterion if $q|2^{n-1}n$.

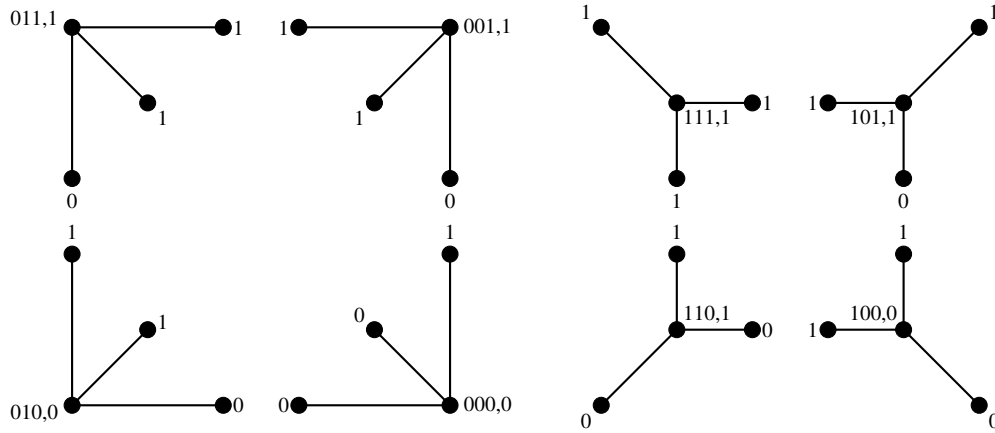
Proof. Let $f \in \mathcal{GB}_n^q$ be a PSAC generalized Boolean function. Let $H_n = (\mathbb{V}_n, E)$ be the labeled hypercube corresponding to f , where \mathbb{V}_n and E are the respective vertex and edge sets of H_n . Let each vertex $\mathbf{x} \in \mathbb{V}_n$ be labeled with an output from \mathbb{Z}_q and let $\lambda(\mathbf{x})$ be the function which returns the label for \mathbf{x} . Moreover, let each edge $\varepsilon = \{\mathbf{x}, \mathbf{y}\} \in E$, $\mathbf{x}, \mathbf{y} \in \mathbb{V}_n$, be labeled with a value $v \in \mathbb{F}_2$, such that $v = \delta(\lambda(\mathbf{x}), \lambda(\mathbf{y}))$, where δ is the Kronecker delta function. By Definition 4.12, in order for f to be PSAC, it must remain invariant with probability $1/q$ for the set of 2^{n-1} possible input vectors. There are a total of $2^{n-1}n$ edges in H_n . Consequently, if f is PSAC, it means that $(2^{n-1}n)/q$ of the edges of H_n must be labeled with 1's. This in turn can only occur if $q|2^{n-1}n$. ■

4.4 Global and Uniform Avalanche Criteria

From a probabilistic frame of reference several types of strict avalanche criteria exist. To illustrate the concept, consider the following labeled H_3 hypercube which represents a SAC Boolean functions $f \in \mathcal{B}_3$:



For each vertex in the graph, we compare its label to the set of labels of its neighbor vertices. For the benefit of the reader, we split H_3 into subgraphs and omit vertex labels other than the one under consideration.

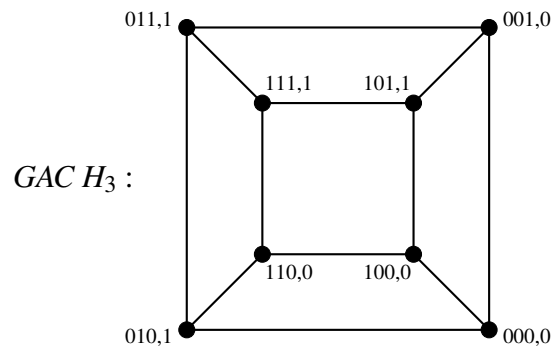


For each vertex we now compute the probability associated with the label remaining invariant as we move from the vertex to its neighbors. The results of these calculations have been compiled in Table 4.3.

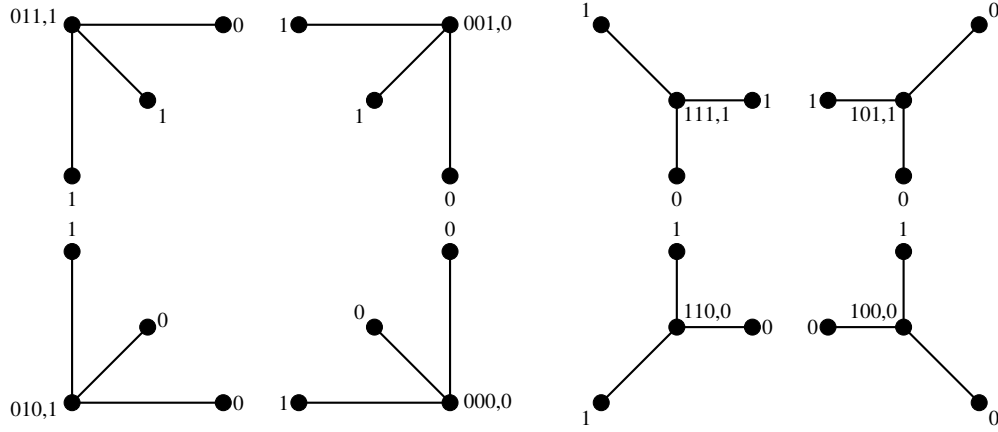
Table 4.3: Vertex invariance probability for a *SAC* Boolean function

<i>Vertex</i>	<i>Prob. Invariance</i>	<i>Prob. Change</i>
000	2/3	1/3
001	2/3	1/3
010	1/3	2/3
011	2/3	1/3
100	1/3	2/3
101	2/3	1/3
110	1/3	2/3
111	1	0

The hypercube is of dimension 3, and each vertex is thus of degree 3. This in turn means that it is impossible to achieve locally balanced invariance and change probabilities at the vertex level. However, summing the respective columns of the table one observes that across the set of all vertices, the probability of invariance exceeds that of the probability of change. From a cryptographic perspective this is an undesirable property! Consider instead the following labeled H_3 hypercube which also represents a *SAC* Boolean function $f \in \mathcal{B}_3$:



To aid the reader we again split H_3 into subgraphs:



As before we calculate the probability of invariance for each vertex of the graph and display the results in Table 4.4.

Table 4.4: Vertex invariance probability for a globally SAC Boolean function

<i>Vertex</i>	<i>Prob. Invariance</i>	<i>Prob. Change</i>
000	2/3	1/3
001	1/3	2/3
010	1/3	2/3
011	2/3	1/3
100	2/3	1/3
101	1/3	2/3
110	1/3	2/3
111	2/3	1/3

Inspecting the results in the Table 4.4, we see that for this SAC hypercube and its associated function the probabilities of invariance and change are balanced across the set of input vectors.

Definition 4.18. A generalized Boolean function $f \in \mathcal{GB}_n^q$ is said to satisfy the *global avalanche criterion (GAC)*, if it satisfies the probabilistic strict avalanche criterion of Defi-

nition 4.12 and,

$$\sum_{\mathbf{x} \in \mathbb{V}_n} \sum_{i=1}^n Pr(f(\mathbf{x} \oplus e_i) = f(\mathbf{x})) = 2^n/q,$$

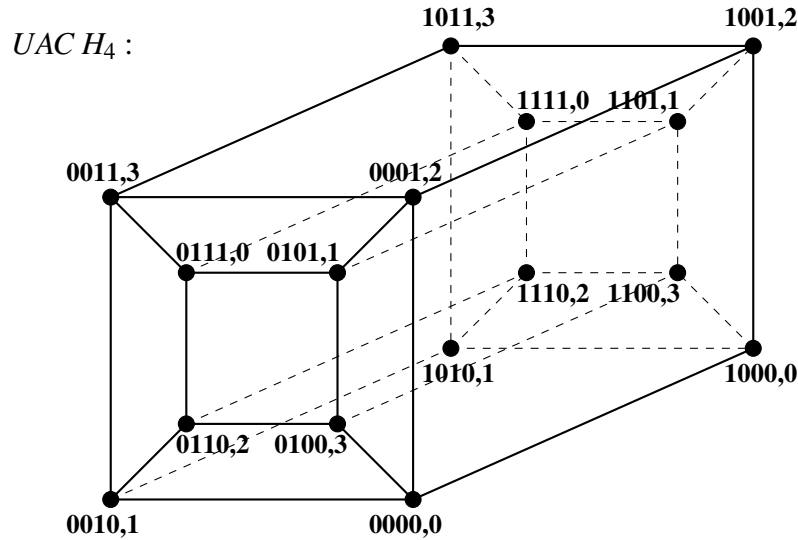
where e_i is a unit vector with the i^{th} bit equal to 1 and all other bits 0.

Definition 4.19. A generalized Boolean function $f \in \mathcal{GB}_n^q$ is said to satisfy the *uniform avalanche criterion (UAC)*, if for all $1 \leq i \leq n$, $1 \leq j \leq q$, and $\mathbf{x} \in \mathbb{V}_n$,

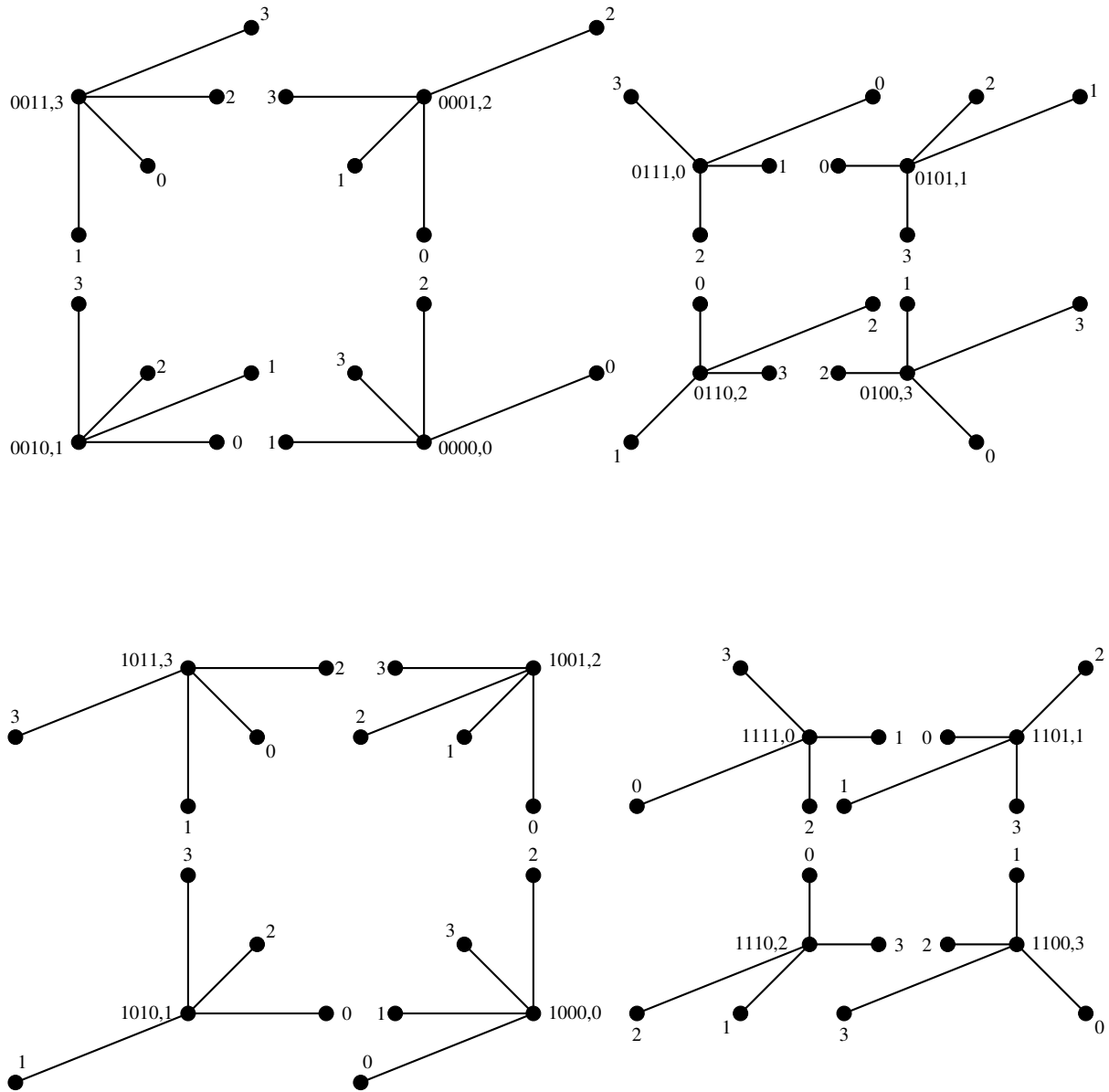
$$Pr(f(\mathbf{x} \oplus e_i) = c_j) = \frac{1}{q},$$

where c_j are distinct elements of \mathbb{Z}_q and e_i are unit vectors with the i^{th} bit equal to 1 and all other bits 0.

Example 4.20. To further motivate the concept of the uniform avalanche criterion, we display the following quaternary output labeled H_4 , which represents a *UAC* generalized Boolean function $f \in \mathcal{GB}_4^4$. For lucidity's sake, we again omit the edge labels.



To help the reader verify that for each vertex in the graph, the set of its neighbors take on all possible output values (labels) from \mathbb{Z}_4 (with equal frequency), we split the graph into 16 subgraphs, one for each of the 16 vertices in H_4 .



The graph paradigm under which we have been operating makes it easy (at least for small examples) to verify that a function satisfies the various avalanche criteria. However, other points of reference also have utility. Consider Table 4.5 which depicts the *UAC* function from Example 4.20.

Table 4.5: A *UAC* generalized Boolean function $f \in \mathcal{GB}_4^4$

\mathbb{V}_4	f
0000	0
0001	2
0010	1
0011	3
0100	3
0101	1
0110	2
0111	0
1000	0
1001	2
1010	1
1011	3
1100	3
1101	1
1110	2
1111	0

From the symmetry exhibited in the first and second half of the truth table, it is apparent that f also is a concatenation of two correlation immune (order 1) generalized Boolean functions (Siegenthaler construction). The fact that this *UAC* function is 1-resilient (*CI*(1) and balanced) is not a coincidence! Generalized Boolean functions which satisfy the uniform avalanche criterion exhibit amazing properties. We will continue to explore these properties throughout the remainder of this chapter.

Theorem 4.21. *If a generalized Boolean function in \mathcal{GB}_n^q satisfies the uniform avalanche criterion, then $q = 2^\ell$, where $\ell \leq n - 1$ if n odd, or $\ell \leq n$, if n even.*

Proof. Let $f \in \mathcal{GB}_n^q$ be a *UAC* generalized Boolean function. Let $H_n = (\mathbb{V}_n, E)$ be the labeled hypercube corresponding to f , where \mathbb{V}_n and E are the respective vertex and edge sets of H_n . Let each vertex $\mathbf{x} \in \mathbb{V}_n$ be labeled with an output from \mathbb{Z}_q . Additionally let

$e_i \in \mathbb{V}_n$ be unit vectors with the i^{th} bit equal to 1 and all other bits 0. By Definition 4.19, in order for f to be UAC, for all $i = 1$ to n , $j = 1$ to q , and $\mathbf{x} \in \mathbb{V}_n$, $Pr(f(\mathbf{x} \oplus e_i) = c_j) = 1/q$. Consequently, not only must the number of edges in the graph, namely $2^{n-1}n$, be divisible by q , but the number of graph vertices must also be divisible by q . A hypercube H_n contains 2^n vertices. Hence, the stipulated requirement has been proven. ■

4.5 Necessary and Sufficient Conditions for a Generalized Strict Avalanche Criterion

Suppose that we wish to employ two generalized Boolean function $f_1 \in \mathcal{GB}_n^{q_1}$ and $f_2 \in \mathcal{GB}_n^{q_2}$ as S-box components of a cryptographic system, as depicted in Figure 4.5. Let S be the $q_1 \times q_2$ S-box (two dimensional array) containing q_1 rows and q_2 columns of binary vector elements of length n . Let $\mathbf{x}, \mathbf{y} \in \mathbb{V}_n$ and given \mathbf{x} , let $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ be the respective row and column pointers into S , such that $f_1(\mathbf{x}) \in \mathbb{Z}_{q_1}$ and $f_2(\mathbf{x}) \in \mathbb{Z}_{q_2}$ and $g(\mathbf{x}) = S[f_1(\mathbf{x})][f_2(\mathbf{x})] = \mathbf{y}$, is the function which returns element \mathbf{y} located in row $f_1(\mathbf{x})$ and column $f_2(\mathbf{x})$ of the S-box.

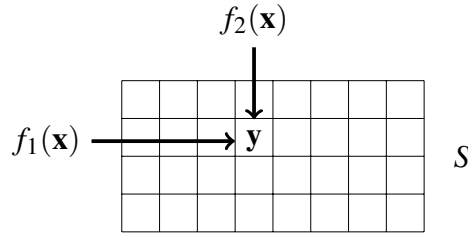
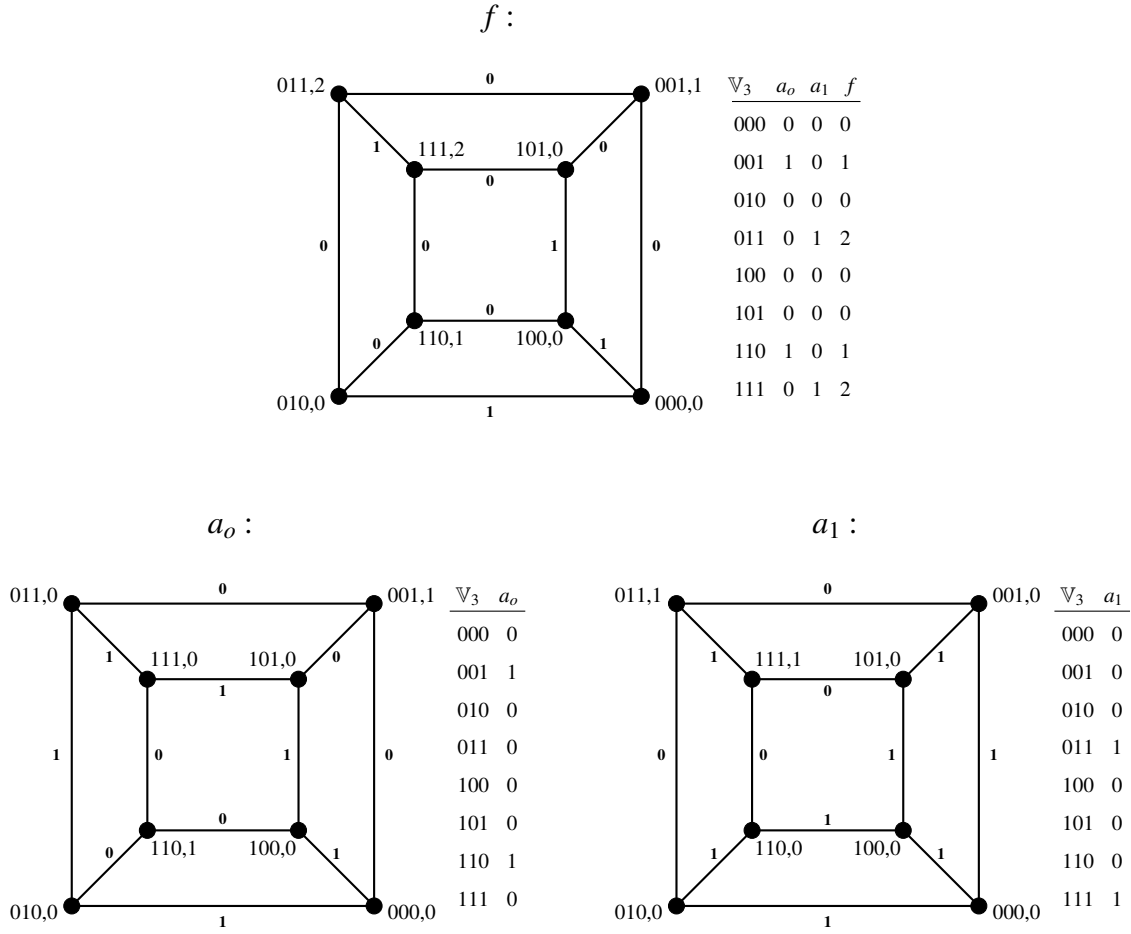


Figure 4.1: S-box using generalized Boolean function pointers

Momentarily considering the q -ary nature of the S-box pointers, one realizes, that in order for the S-box in question to exhibit good cryptographic properties, it is imperative that in addition to f_1 and f_2 being PSAC, each of their constituent Boolean functions must also be SAC. Regrettably, unlike the situation encountered for correlation immunity, the fact that a generalized Boolean function is PSAC does not guarantee that its Boolean function components will also be SAC.

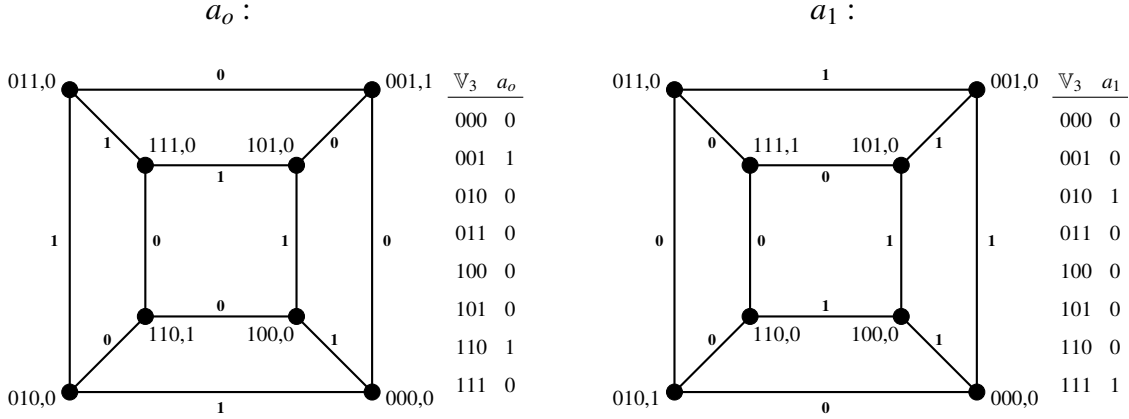
Example 4.22. To see that this is the case, consider the following generalized Boolean function $f \in \mathcal{GB}_3^3$ along with its constituent Boolean functions, a_0 and a_1 .



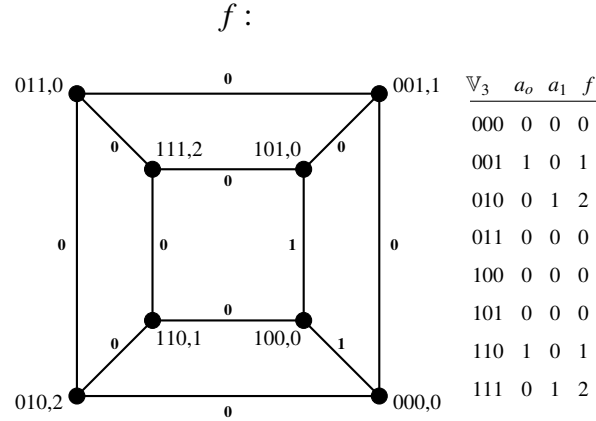
By inspection we see that 4 of the 12 edges of f 's graph are labeled with 1's. The probability that two of neighboring vertices agree with respect to their output values (labels) is therefore $1/3$, so f is *PSAC*. Likewise, 6 of a_0 's 12 edges are labeled with 1's, so it is *SAC*. However, in the case of a_1 , 8 of its 12 edges are labeled with 1's and it therefore fails to satisfy the *SAC*.

Proceeding in the opposite direction and building a generalized Boolean function using *SAC* Boolean functions also does not guarantee that the generalized Boolean function will be *PSAC*. We again provide the reader with an example:

Example 4.23. Start with the following graphs:



Of the 12 edges in each of the labeled graphs a_0 and a_1 , 6 edges are 1-labeled, hence the Boolean functions which they represent are both *SAC*. We now utilize these functions to produce the following generalized Boolean function $f(\mathbf{x}) = a_0(\mathbf{x}) + 2a_1(\mathbf{x})$.



The graph f contains 2 1-labeled edges, which means that the probability of two neighbor vertices in the graph having the same output label is $2/12 = 1/6$. Given the fact that $q = 3$, we conclude that f does not satisfy the *PSAC*.

Both of these situations are unfortunate! Webster and Tavares' notion of a strict avalanche criterion was born out of a desire to build S-boxes with good cryptographic properties. If we hope to employ generalized Boolean functions as components of cryptographic algorithms (quantum, perhaps) we must at minimum avoid introducing binary decomposition

design weaknesses and thus must ensure that the constituent Boolean functions of a *PSAC* generalized Boolean function are all *SAC*. From a practical perspective we would also like to be able to build *PSAC* generalized Boolean functions using *SAC* Boolean function. Bearing both conditions in mind, we formulate the following definition.

Definition 4.24. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function, such that for $\mathbf{x} \in \mathbb{V}_n$, $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, $a_j \in \mathcal{B}_n$. The function f is said to satisfy the *generalized strict avalanche criterion (GSAC)* if and only if f satisfies the probabilistic strict avalanche criterion and all Boolean functions a_j , $0 \leq j \leq k-1$, satisfy the strict avalanche criterion.

Lemma 4.25. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function, such that $\mathbf{x} \in \mathbb{V}_n$ and $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, where $a_j \in \mathcal{B}_n$. If f satisfies the uniform avalanche criterion, then for all j , $0 \leq j \leq k-1$, a_j satisfies the strict avalanche criterion.

Proof. Let $f \in \mathcal{GB}_n^q$ be a *UAC* generalized Boolean function. Let $H_n = (\mathbb{V}_n, E)$ be the labeled hypercube corresponding to f , where \mathbb{V}_n and E are the respective vertex and edge sets of H_n . Let each vertex $\mathbf{x} \in \mathbb{V}_n$ be labeled with an output $c_m \in \mathbb{Z}_q$ and let $e_i \in \mathbb{V}_n$ be a unit vector with the i^{th} bit equal to 1 and all other bits 0. By Definition 4.19, in order for f to be *UAC*, for all $i = 1$ to n , all $m = 1$ to q , and every $\mathbf{x} \in \mathbb{V}_n$, $Pr(f(\mathbf{x} \oplus e_i) = c_m) = 1/q$. Since H_n is a hypercube, each vertex is of degree $n = hq$, for some h , $1 \leq h \leq n$. Moreover, from Theorem 4.21, we know that $q = 2^\ell$ for $\ell \leq n$. For each value j , $j = 0$ to $k-1$, and each vertex \mathbf{x} , we relabel H_n by replacing the output value (label) c_m with the j^{th} bit of the binary expansion of c_m , thus creating a new labeled hypercube for each Boolean function a_j . Consider further the binary expansion of the set of q distinct output values $c_m \in \mathbb{Z}_q$. Observe that since $q = 2^\ell$, for each j this set will contain an equal number of 0's and 1's. If this is not immediately evident, consider the fact that each column j of \mathbb{V}_ℓ is balanced. Since f is *UAC*, for each vertex \mathbf{x} in H_n , each value q appears with frequency h in the set of neighbor vertices of \mathbf{x} . Therefore, regardless of what value h happens to be for our particular generalized Boolean function f , for each Boolean function, a_j , each vertex \mathbf{x} in a_j will have $2^{\ell-1}$ neighbor vertices with 0 labels and $2^{\ell-1}$ neighbor vertices with 1 labels. Hence a_j satisfies the uniform avalanche criterion and thus is also *SAC*. ■

Lemma 4.26. Let $B = \{a_0, a_1, \dots, a_{k-1}\}$ be a set of k Boolean functions each in n variables. If each Boolean function satisfies the uniform avalanche criterion (*UAC*) and for all j and

h , where $0 \leq j, h, \leq k-1$ and $j \neq h$, the pairwise Hamming distance $d(a_j, a_h) = 2^{n-1}$, then the generalized Boolean function $f \in \mathcal{GB}_n^q$, constructed using B such that $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, will be such that it also satisfies the uniform avalanche criterion.

Proof. Let $B = \{a_0, a_1, \dots, a_{k-1}\}$ be a set of k UAC Boolean functions each in n variables. For all j and h , where $0 \leq j, h, \leq k-1$ and $j \neq h$, let each function be such that their pairwise Hamming distances satisfy $d(a_j, a_h) = 2^{n-1}$. Let $f \in \mathcal{GB}_n^q$ be the generalized Boolean function constructed using B such that $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$. For $i = 1$ to n , let, $V_{\mathbf{x}} = \{\mathbf{x} \oplus e_i : \mathbf{x} \in \mathbb{V}_n\}$, be the set of vectors of Hamming distance 1 from \mathbf{x} , and denote $C_{\mathbf{x}} = f(V_{\mathbf{x}})$ as the set of output values associated with $V_{\mathbf{x}}$. Consider now the q distinct output values $c_m \in \mathbb{Z}_q$, $m = 1$ to q . Indexing from $j = 0$ to $k-1$, let $(c_m)_2(j)$ represent the j^{th} bit of the binary expansion of c_m . Each Boolean function is UAC, therefore for all $i = 1$ to n , all $m = 1$ to q , every position j , and all fixed $\mathbf{x}'s$, $Pr(a_j(\mathbf{x} \oplus e_i) = (c_m)_2(j)) = 1/2$. In other words, the number of 0's and 1's are equal for each index j , of the set of vectors $C_{\mathbf{x}}$. Moreover, since the pairwise Hamming distance between all distinct Boolean functions is 2^{n-1} , it means that the q output values of $C_{\mathbf{x}}$ will all be distinct elements of \mathbb{V}_k . Thus, it must be the case that for all $\mathbf{x} \in \mathbb{V}_n$, $Pr(f(\mathbf{x} \oplus e_i) = c_m) = 1/q$ proving that f is UAC. ■

Theorem 4.27. A generalized Boolean function $f \in \mathcal{GB}_n^q$, $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, where $\mathbf{x} \in \mathbb{V}_n$ and $a_j \in \mathcal{B}_n$, is GSAC if f and all functions a_j are UAC and for all $0 \leq j, h \leq k-1$, such that $j \neq h$, the pairwise Hamming distance $d(a_j, a_h) = 2^{n-1}$.

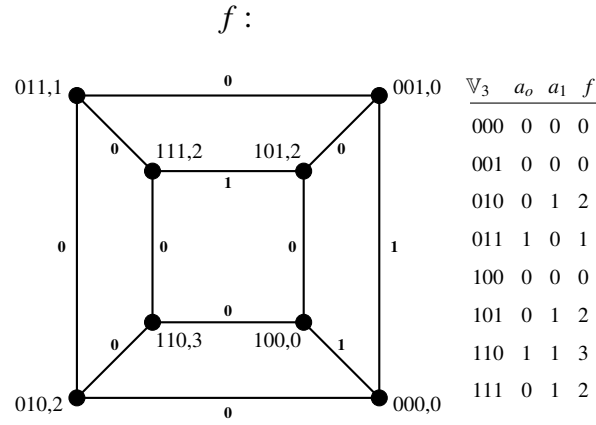
Proof. According to Definition 4.24, a generalized Boolean function $f \in \mathcal{GB}_n^q$, where $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, satisfies the generalized strict avalanche criterion if and only if f satisfies the probabilistic strict avalanche criterion and all Boolean functions a_j , $j = 0$ to $k-1$, satisfy the strict avalanche criterion.

(\Rightarrow) Let $f \in \mathcal{GB}_n^q$ be a UAC generalized Boolean function such that $\mathbf{x} \in \mathbb{V}_n$, $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$, and $a_j \in \mathcal{B}_n$. Then according to Lemma 4.25, all Boolean functions a_j are SAC.

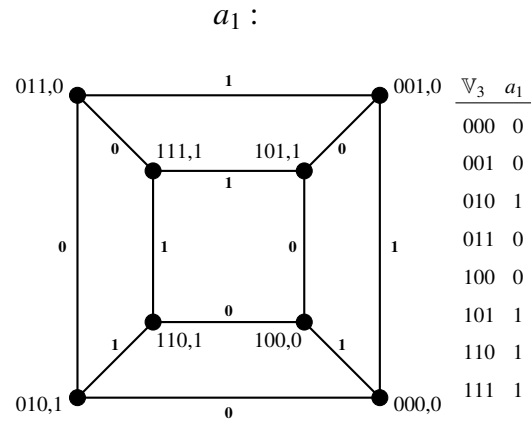
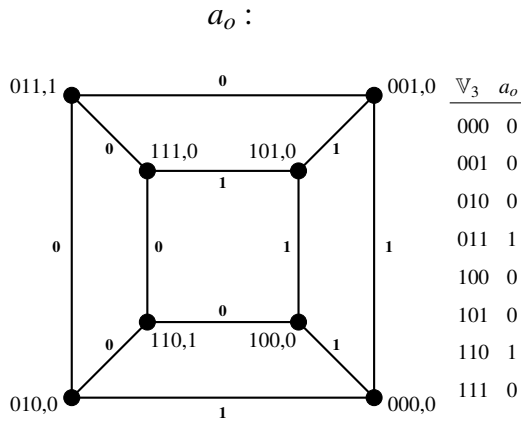
(\Leftarrow) Let $B = \{a_0, a_1, \dots, a_{k-1}\}$ be a set of k Boolean functions, each in n variables and each of which also satisfy the uniform avalanche criterion. For all j and h , where $0 \leq j, h, \leq k-1$ and $j \neq h$, let the pairwise Hamming distance $d(a_j, a_h) = 2^{n-1}$. Suppose $f \in \mathcal{GB}_n^q$, is a

generalized Boolean function constructed using B such that $f(\mathbf{x}) = \sum_{j=0}^{k-1} 2^j a_j(\mathbf{x})$. Then according to Lemma 4.26, f satisfies the uniform avalanche criterion. ■

Examples of *GSAC* generalized Boolean functions abound. The *UAC* generalized Boolean function $f \in \mathcal{GB}_4^4$ which we presented in Example 4.20 satisfied the generalized strict avalanche criterion. Below we provide yet another example of a generalized Boolean function $f \in \mathcal{GB}_3^4$ which satisfies the generalized strict avalanche criterion. In this case however, the function fails to satisfy the *UAC*.



Observe that 3 of the 12 edges in the graph f are 1-labeled. The probability that any two neighbor vertices in the graph have the same output value (label) is therefore 1/4 and the function is *PSAC*.



In a_0 and a_1 , 6 of the 12 graph edges in each respective graph are 1-labeled. Thus, the probability that any two neighbor vertices in either graph having the same output value (label) is therefore $1/2$ and both functions are therefore SAC.

4.6 The Connection between the Uniform Avalanche Criterion and Correlation Immunity

In Example 4.20 we hinted that a connection existed between a function satisfying the uniform avalanche criterion and the fact that it was correlation immune (order 1). We now prove this result.

Theorem 4.28. *Generalized Boolean functions $f \in \mathcal{GB}_n^q$ which satisfy the uniform avalanche criterion are 1-resilient (balanced and correlation immune of order 1).*

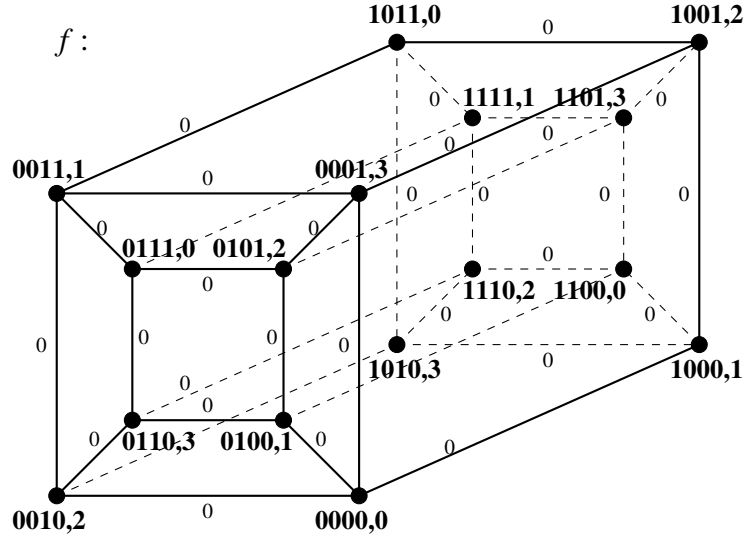
Proof. We proceed by way of contradiction. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function which satisfies the uniform avalanche criterion. Partition the set of input vectors \mathbb{V}_n into q sets X_j , where $0 \leq j \leq q-1$, such that for all $\mathbf{x} \in X_j$, $f(\mathbf{x}) = j$. Without loss of generality consider one of these sets X_j , say for instance X_0 . Suppose that there exists at least one index k , $1 \leq k \leq n$ for which the set of vectors X_0 , contain an uneven number of 0's and 1's. Let e_i denote a unit vector with the i^{th} bit equal to 1 and all other bits 0. The function f is UAC, so for the set of unit vectors, where $i = 1$ to n and each $\mathbf{x} \in X_0$, the vectors, $\mathbf{x} \oplus e_i$, each reside in one of the q different sets X_j . Therefore any imbalance with respect to the number of 0's and 1's in column k for the vectors of X_0 must also result in a 0-1 imbalance in column k of the vectors contained in each of the $q-1$ remaining sets X_j , where $j \neq 0$. Assume that there is a difference of d more 0's than 1's in column k of X_0 . Since f is UAC, the total disparity of 0's and 1's for all vectors in the remaining sets X_j , $1 \leq j \leq q-1$ is $d(n-1)$. However, the union $\cup_{j=0}^{q-1} X_j = \mathbb{V}_n$. Since the number of 0's and 1's is balanced for each column i , $i = 1$ to n this cannot occur. We therefore conclude that for all indices $i = 1$ to n and each set X_j , $j = 0$ to $q-1$, there must be an equal number of 0's and 1's. This in turn means that for all $j \in \mathbb{Z}_q$ and every i from 1 to n , $\Pr(x_i = 1 | f(\mathbf{x}) = j) = 1/2$, which implies that f is CI(1). Moreover, f is UAC, so for each \mathbf{x} , and each $c_m \in \mathbb{Z}_q$, $\Pr(f(\mathbf{x} \oplus e_i) = c_m) = 1/q$. Thus each output value c_m occurs with equal frequency across all $\mathbf{x} \in \mathbb{V}_n$ and f is therefore balanced. \blacksquare

Remark 4.29. Theorem 4.28 is important. It not only tells us that a *UAC* generalized Boolean function is also *CI*(1), but given Theorems 3.38 and 4.25, also says that the constituent Boolean function from which f was built are all *CI*(1) and *SAC*, thus rendering f resistant to the binary decomposition attacks, which we previously considered. Notice, however, that although all generalized Boolean functions which satisfy the uniform avalanche criterion are correlation immune (order 1), not all order-1 correlation immune generalized Boolean functions are *UAC*, or even *PSAC*, for that matter.

Example 4.30. To see that this is the case, consider the (order 1) correlation immune generalized Boolean function $f \in \mathcal{GB}_4^4$ in Table 4.6 along with its associated labeled hypercube.

Table 4.6: A *non-UAC CI*(1) generalized Boolean function $f \in \mathcal{GB}_4^4$

\mathbb{V}_4	f
0000	0
0001	3
0010	2
0011	1
0100	1
0101	2
0110	3
0111	0
1000	1
1001	2
1010	3
1011	0
1100	0
1101	3
1110	2
1111	1



Using symmetry as our aid, we clearly see that f is a $CI(1)$ generalized Boolean function. However, in this extreme case, none of the 32 edges in the corresponding graph are 1-labeled. Thus f not only fails to satisfy the UAC , but also fails to be $PSAC$.

Using two UAC compliant generalized Boolean functions in n variables along with Algorithm 7 and the Siegenthaler construction allows us to construct a generalized Boolean functions in $n + 1$ variables which is both $PSAC$ and 1-resilient.

Example 4.31. Using the two UAC generalized Boolean functions in Tables 4.7 and 4.8 along with Algorithm 7 and the Siegenthaler construction we construct the $PSAC$ and 1-resilient function depicted in Table 4.9 and Figure 4.2.

Table 4.7: *UAC* function $f_1 \in \mathcal{GB}_4^4$

\mathbb{V}_4	a_0	a_1	$a_0 \oplus a_1$	f_1
0000	0	0	0	0
0001	0	1	1	2
0010	1	0	1	1
0011	1	1	0	3
0100	1	1	0	3
0101	1	0	1	1
0110	0	1	1	2
0111	0	0	0	0
1000	0	0	0	0
1001	0	1	1	2
1010	1	0	1	1
1011	1	1	0	3
1100	1	1	0	3
1101	1	0	1	1
1110	0	1	1	2
1111	0	0	0	0

Table 4.8: *UAC* function $f_2 \in \mathcal{GB}_4^4$

\mathbb{V}_4	a_0	a_1	$a_0 \oplus a_1$	f_2
0000	0	0	0	0
0001	1	1	0	3
0010	0	1	1	2
0011	1	0	1	1
0100	1	0	1	1
0101	0	1	1	2
0110	1	1	0	3
0111	0	0	0	0
1000	0	0	0	0
1001	1	1	0	3
1010	0	1	1	2
1011	1	0	1	1
1100	1	0	1	1
1101	0	1	1	2
1110	1	1	0	3
1111	0	0	0	0

Table 4.9: A *PSAC* and 1-resilient generalized Boolean function $f_1 \parallel f_2 = f \in \mathcal{GB}_5^4$

\mathbb{V}_4	a_0	a_1	$a_0 \oplus a_1$	f
00000	0	0	0	0
00001	0	1	1	2
00010	1	0	1	1
00011	1	1	0	3
00100	1	1	0	3
00101	1	0	1	1
00110	0	1	1	2
00111	0	0	0	0
01000	0	0	0	0
01001	0	1	1	2
01010	1	0	1	1
01011	1	1	0	3
01100	1	1	0	3
01101	1	0	1	1
01110	0	1	1	2
01111	0	0	0	0
10000	0	0	0	0
10001	1	1	0	3
10010	0	1	1	2
10011	1	0	1	1
10100	1	0	1	1
10101	0	1	1	2
10110	1	1	0	3
10111	0	0	0	0
11000	0	0	0	0
11001	1	1	0	3
11010	0	1	1	2
11011	1	0	1	1
11100	1	0	1	1
11101	0	1	1	2
11110	1	1	0	3
11111	0	0	0	0

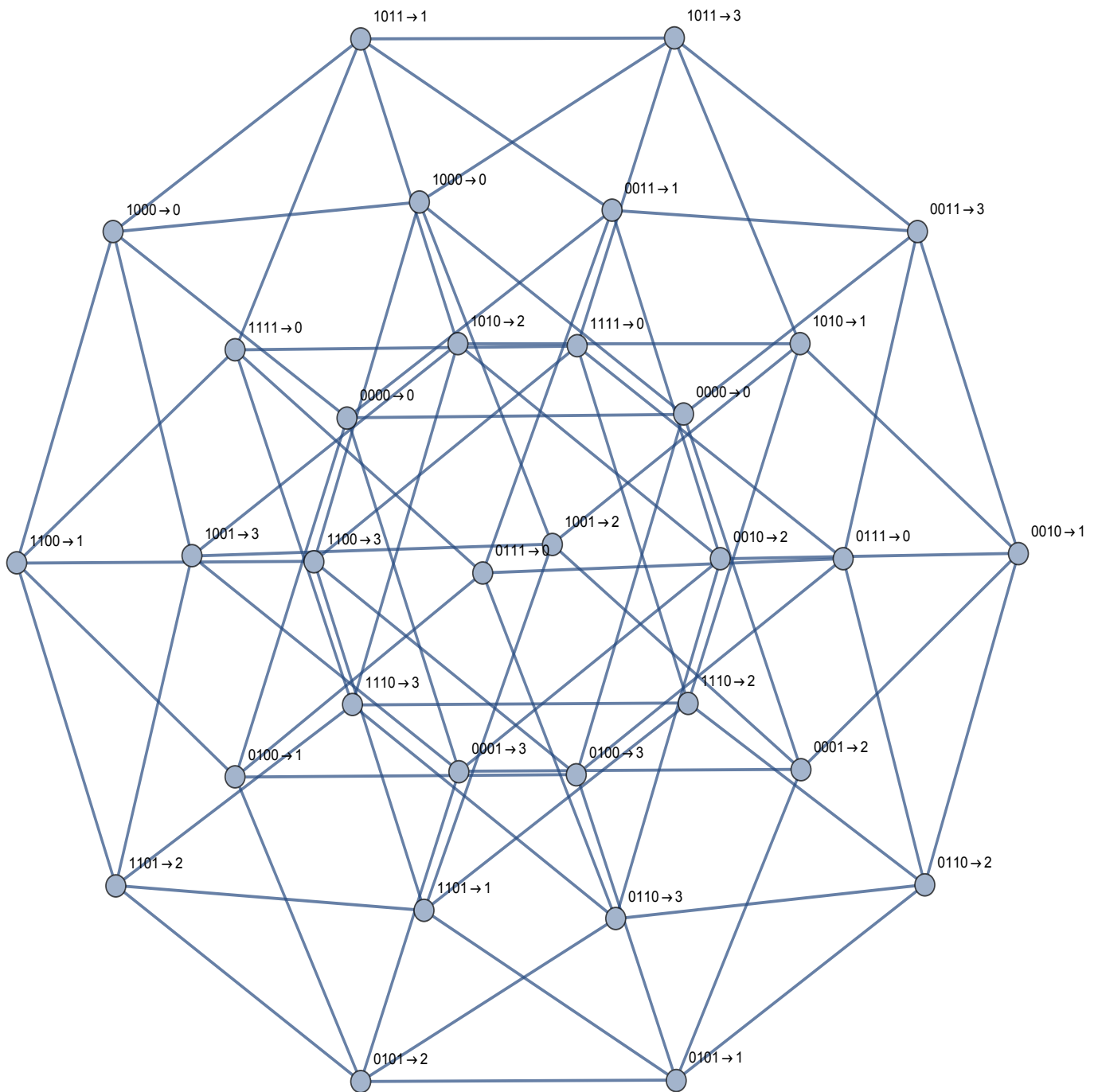


Figure 4.2: Labeled hypercube corresponding to the generalized Boolean function in Table 4.9

4.7 Linear Structures and the Globally Uniform Gradient

The preceding discourse on strict avalanche criteria prompted us to examine the behavior of a generalized function, first across the entire set of input vectors, and later for each individual input vector. By proceeding from the "global" to "local" point of view, and along the way modifying requirements so as to ensure that output value probabilities remained balanced, we were able to devise increasingly well-behaved functions. The pinnacle of our analysis thus far has been the set of functions which satisfy the uniform strict avalanche criterion. These functions are both 1-resilient and satisfy the generalized strict avalanche criterion. However, more remains to be done.

Recall from Definition 2.16 that, given a generalized Boolean function $f \in \mathcal{GB}_n^q$, a vector $\mathbf{a} \in \mathbb{V}_n$ is called a linear structure if there exists $c \in \mathbb{Z}_q$ such that, for all $\mathbf{x} \in \mathbb{V}_n$, $f(\mathbf{x} \oplus \mathbf{a}) - f(\mathbf{x}) = c$.

Consider once again the function f_1 from Example 4.31. We partition the input vectors X_j , $j = 0$ to 3, such that $\cup_{j=0}^3 X_j = \mathbb{V}_4$ and for all $\mathbf{x} \in X_j$, $f_1(\mathbf{x}) = j$, where $j \in \mathbb{Z}_4$:

X_0
0000
1000
0111
1111

X_1
0010
1010
0101
1101

X_2
0001
1001
0110
1110

X_3
0011
1011
0100
1100

Let $e_4 = 1000$ and observe that for each set X_j and for all $\mathbf{x} \in X_j$, $f_1(\mathbf{x}) = f_1(\mathbf{x} \oplus e_4)$. Thus, e_4 is a linear structure and the output invariance for f is skewed in the direction of e_4 . From a cryptographer's standpoint this is undesirable! The weakness in f_1 stems from the way it was constructed. Concatenating two identical copies of a generalized Boolean function $g \in \mathcal{GB}_n^q$ will always introduce the linear structure e_n into the newly constructed function. While the ease of such a construction may be tempting, it, like so many things in cryptography, comes with trade-offs. Consider on the other hand the generalized Boolean function in Table 4.10, which also happens to satisfy the UAC.

Table 4.10: A UAC function $f \in \mathcal{GB}_4^4$, without e_i as a linear structure

\mathbb{V}_4	a_0	a_1	$a_0 \oplus a_1$	f
0000	1	0	1	1
0001	1	0	1	1
0010	1	1	0	3
0011	0	1	1	2
0100	0	0	0	0
0101	1	1	0	3
0110	0	0	0	0
0111	0	1	1	2
1000	0	1	1	2
1001	0	0	0	0
1010	1	1	0	3
1011	0	0	0	0
1100	0	1	1	2
1101	1	1	0	3
1110	1	0	1	1
1111	1	0	1	1

Indexing from right to left and $i = 1$ to n , let e_i be the unit vector with 1 in i^{th} position and 0 everywhere else. Once again, we partition the input vectors X_j , $j = 0$ to 3, such that $\cup_{j=0}^3 X_j = \mathbb{V}_4$ and for all $\mathbf{x} \in X_j$, $f(\mathbf{x}) = j$, where $j \in \mathbb{Z}_4$.

X_0
0100
0110
1001
1011

X_1
0000
0001
1110
1111

X_2
0011
0111
1000
1100

X_3
0010
1010
0101
1101

Using this partition, we subsequently consider which unit vectors result in invariance among the output values for f . Doing so we discover the following:

- For all $\mathbf{w} \in X_0$, $f(\mathbf{w}) = f(\mathbf{w} \oplus e_2)$
- For all $\mathbf{x} \in X_1$, $f(\mathbf{x}) = f(\mathbf{x} \oplus e_1)$
- For all $\mathbf{y} \in X_2$, $f(\mathbf{y}) = f(\mathbf{y} \oplus e_3)$
- For all $\mathbf{z} \in X_3$, $f(\mathbf{z}) = f(\mathbf{z} \oplus e_4)$.

This situation is much improved! Now each unit vector is associated with one of the 4 sets of the partition.

A considerable amount of effort has thus far gone into designing generalized Boolean functions $f \in \mathcal{GB}_n^q$ such that, for each $\mathbf{x} \in \mathbb{V}_n$ and all i from 1 to n , the function ensures that for the set of all Hamming distance 1 vectors, f achieves all output values in \mathbb{Z}_q with equal probability. It therefore only seems natural that we also ensure that for each $\mathbf{x} \in \mathbb{V}_n$, the probability $Pr(f(\mathbf{x}) = f(\mathbf{x} \oplus e_i))$ is equal for each of the n unit vectors in f .

Definition 4.32. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function which satisfies the uniform avalanche criterion and let e_i denote a unit vector with the i^{th} bit equal to 1 and all other bits 0. The function f is said to possess a globally uniform gradient if for each e_i , $1 \leq i \leq n$,

$$Pr(D_{e_i}f(\mathbf{x}) = 0) = \frac{1}{n},$$

where $D_{e_i}f(\mathbf{x}) = f(\mathbf{x} \oplus e_i) - f(\mathbf{x})$, is the derivative of f with respect to the unit vector e_i . Generalized Boolean functions which satisfy the UAC and have a globally uniform unit vector gradient are referred to as **Cataract** functions.

Definition 4.33. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function and let e_i denote a unit vector with the i^{th} bit equal to 1 and all other bits 0. Then for all $\mathbf{x} \in \mathbb{V}_n$ and $i = 1$ to n , we define the gradient of f , denoted $\nabla f_{e_i}(\mathbf{x})$, as follows:

$$\nabla f_{e_i}(\mathbf{x}) = \langle D_{e_1}f(\mathbf{x}), D_{e_2}f(\mathbf{x}), \dots, D_{e_n}f(\mathbf{x}) \rangle,$$

where $D_{e_i}f(\mathbf{x})$ is the derivative of f with respect to the unit vector e_i .

Theorem 4.34. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function which satisfies the uniform avalanche criterion. Let $\mathbf{x} \in \mathbb{V}_n$ and denote e_i as a unit vector with the i^{th} bit equal to 1 and all other bits 0. Then $\{\nabla f_{e_i}(\mathbb{V}_n)\} = \mathbb{Z}_q, \forall i, 1 \leq i \leq n$.

Proof. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function which satisfies the uniform avalanche criterion. Since f is UAC, for $i = 1$ to n , $f(\mathbf{x} \oplus e_i)$ when \mathbf{x} runs through \mathbb{V}_n , must achieve all values of \mathbb{Z}_q (with equal frequency). Subtraction in the derivative, $D_{e_i}f(\mathbf{x})$, is carried modulo q , thus, for each distinct i , $f(\mathbf{x} \oplus e_i) - f(\mathbf{x})$ is a unique element of \mathbb{Z}_q . ■

Theorem 4.35. *Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function. Let $\mathbf{x} \in \mathbb{V}_n$ and denote e_i as a unit vector with the i^{th} bit equal to 1 and all other bits 0. If f satisfies the uniform avalanche criterion and has a globally uniform gradient, then for all $\mathbf{x} \in \mathbb{V}_n$, and for specific i , the set $\{D_{e_i}f(\mathbf{x})\}$ contains all elements of \mathbb{Z}_q in balanced proportions (in other words, it is a permutation of the truth table of f).*

Proof. Let $f \in \mathcal{GB}_n^q$ be a generalized Boolean function which satisfies the uniform avalanche criterion and which has a globally uniform gradient. The function f has a globally uniform gradient, thus according to Definition 4.32, for each specific i and unit vector e_i , there are $2^n/n$ vectors $\mathbf{x} \in \mathbb{V}_n$ for which $D_{e_i}f(\mathbf{x}) = 0$. However, f is also UAC, so according to Theorem 4.34, for each \mathbf{x} and all i from 1 to n , $\{\nabla f_{e_i}(\mathbf{x})\} = \mathbb{Z}_q$. Thus, in order for both conditions to hold, it must be the case that for each specific unit vector, e_i , and the set of all vectors \mathbb{V}_n , each value $D_{e_i}f(\mathbf{x}) \in \mathbb{Z}_q$ occurs with frequency a divisor of 2^n . ■

We can use Theorem 4.35 to evaluate whether or not a generalized Boolean function that satisfies the uniform avalanche criterion also has a globally uniform gradient. We demonstrate the approach using the following example.

Example 4.36. Suppose we would like to check whether or not the functions f_1 and f_2 from our previous example each satisfy the uniform avalanche criterion and have globally uniform gradients. Using their truth tables, we compute their respective gradients for all vectors $\mathbf{x} \in \mathbb{V}_n$. The results from these calculations are displayed in Table 4.11.

Table 4.11: Gradients for two *UAC* generalized Boolean functions f_1 and f_2

\mathbb{V}_4	f_1	f_2	$\nabla f_{1e_i}(\mathbf{x})$	$\nabla f_{2e_i}(\mathbf{x})$
0000	0	1	$\langle 2, 1, 3, 0 \rangle$	$\langle 0, 2, 3, 1 \rangle$
0001	2	1	$\langle 2, 1, 3, 0 \rangle$	$\langle 0, 1, 2, 3 \rangle$
0010	1	3	$\langle 2, 3, 1, 0 \rangle$	$\langle 3, 2, 1, 0 \rangle$
0011	3	2	$\langle 2, 3, 1, 0 \rangle$	$\langle 1, 3, 0, 2 \rangle$
0100	3	0	$\langle 2, 3, 1, 0 \rangle$	$\langle 3, 0, 1, 2 \rangle$
0101	1	3	$\langle 2, 3, 1, 0 \rangle$	$\langle 1, 3, 2, 0 \rangle$
0110	2	0	$\langle 2, 1, 3, 0 \rangle$	$\langle 2, 0, 3, 1 \rangle$
0111	0	2	$\langle 2, 1, 3, 0 \rangle$	$\langle 2, 1, 0, 3 \rangle$
1000	0	2	$\langle 2, 1, 3, 0 \rangle$	$\langle 2, 1, 0, 3 \rangle$
1001	2	0	$\langle 2, 1, 3, 0 \rangle$	$\langle 2, 0, 3, 1 \rangle$
1010	1	3	$\langle 2, 3, 1, 0 \rangle$	$\langle 1, 3, 2, 0 \rangle$
1011	3	0	$\langle 2, 3, 1, 0 \rangle$	$\langle 3, 0, 1, 2 \rangle$
1100	3	2	$\langle 2, 3, 1, 0 \rangle$	$\langle 1, 3, 0, 2 \rangle$
1101	1	3	$\langle 2, 3, 1, 0 \rangle$	$\langle 3, 2, 1, 0 \rangle$
1110	2	1	$\langle 2, 1, 3, 0 \rangle$	$\langle 0, 1, 2, 3 \rangle$
1111	0	1	$\langle 2, 1, 3, 0 \rangle$	$\langle 0, 2, 3, 1 \rangle$

Examining the rows of the table, for each vector \mathbf{x} , we observe that the gradients for both functions contain all values \mathbb{Z}_q . Turning our attention to the columns of each respective set of gradients, we moreover observe the following: For each column i from 1 to n , the gradient values associated with e_i for f_1 are not balanced. For example, the values in the first column (associated with e_1), are all 2. This however is not the case for f_2 . Here we see that for each column, i , the e_i -associated derivatives in the set of gradients, all appear with equal frequency. We therefore conclude that f_1 does not possess a uniform gradient, whereas f_2 does.

CHAPTER 5:

Generalized Bent Boolean Functions

Mathematics compares the most
diverse phenomena and discovers the
secret analogies that unite them.

Joseph Fourier ✍

This chapter includes results on generalized bent Boolean functions from the following papers: *Bent and generalized bent Boolean functions* [44], *Generalized bent functions and their Gray images* [28], as well as *Partial spread and vectorial generalized bent functions* [29]. The dissertation author is a coauthor on these papers. The discourse along with all results appear in the original form in which they were published in the cited works.

5.1 Introduction

The culmination of our investigation into avalanche features for generalized Boolean functions was the development of what we referred to as *cataract* functions. These functions are *UAC*, free of unit vector linear structures, and contain a global uniform gradient. In this section we expand upon the idea of removing linear structures from a generalized Boolean function. Meier and Staffelbach [30] investigated a class of Boolean functions which they called perfectly nonlinear. We extend here their notion of perfect nonlinear Boolean functions so that it applies to generalized Boolean functions.

Definition 5.1. A generalized Boolean function $f : \mathbb{V}_n \rightarrow \mathbb{Z}_q$ is called *perfect nonlinear* with respect to linear structures (perfect nonlinear for short) if for every $0 \leq j \leq q - 1$, and every nonzero vector $\mathbf{a} \in \mathbb{V}_n$, the equation $D_{\mathbf{a}}f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{a}) - f(\mathbf{x}) = j$ has exactly $2^n/q$ solutions $\mathbf{x} \in \mathbb{V}_n$ (in other words, the derivatives of f at every point \mathbf{a} are balanced).

Remark 5.2. Notice that based on the above definition, in order for a generalized Boolean function $f \in \mathcal{GB}_n^q$ to be perfect nonlinear, q must be such that $q = 2^\ell$, where $1 \leq \ell \leq n - 1$.

In their cited paper, Meier and Staffelbach demonstrated that the class of perfect nonlinear and bent Boolean functions coincide.

Generalized bent Boolean function is an active area research. A plethora of papers have been written on the topic (see [28], [29], [44] and the references therein). We present here a few results contained in the above cited papers which were coauthored by this dissertation author.

5.2 Generalized Bent Boolean Functions

The material presented in this section was taken directly from the paper *Bent and generalized bent Boolean functions* [44] and appears in its original published form.

Recall from Chapter 2 that the generalized *Walsh–Hadamard transform* of $f \in \mathcal{GB}_n^q$ at any point $\mathbf{u} \in \mathbb{V}_n$ is the complex valued function

$$\mathcal{H}_f(\mathbf{u}) = 2^{-\frac{n}{2}} \sum_{\mathbf{x} \in \mathbb{V}_n} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{u} \cdot \mathbf{x}}.$$

Definition 5.3. [44] A function $f \in \mathcal{GB}_n^q$ is a *generalized bent (gbent)* function if $|\mathcal{H}_f(\mathbf{u})| = 1$ for all $\mathbf{u} \in \mathbb{V}_n$. When $q = 2$, then f is bent (these exist for n even, only). If n is odd, a function $f \in \mathcal{B}_n$ is said to be *semibent* if and only if $|W_f(\mathbf{u})| \in \{0, \sqrt{2}\}$, for all $\mathbf{u} \in \mathbb{V}_n$.

Suppose $f \in \mathcal{GB}_n^q$ is a gbent function such that for every \mathbf{u} , we have $\mathcal{H}_f(\mathbf{u}) = \zeta^{k_u}$, for some $0 \leq k_u < q$. Then, for such a gbent function f , there is a function $F : \mathbb{V}_n \rightarrow \mathbb{Z}_q$ such that $\zeta^F = \mathcal{H}_f$. We call such a function F the *dual* of f . The reader is cautioned that only some gbent functions admit duals. By applying Theorem 5.4, one can easily see that the dual of a gbent function is also gbent, since the Walsh–Hadamard transform of the dual F is $\mathcal{H}_F(\mathbf{u}) = \zeta^{f(\mathbf{u})}$ [44].

The following properties of the Walsh–Hadamard transform on generalized Boolean functions are similar to the Boolean function case [44].

Theorem 5.4. [44]

(i) Let $f \in \mathcal{GB}_n^q$. The inverse of the Walsh–Hadamard transform is given by

$$\zeta^{f(\mathbf{y})} = 2^{-\frac{n}{2}} \sum_{\mathbf{u} \in \mathbb{V}_n} \mathcal{H}_f(\mathbf{u})(-1)^{\mathbf{u} \cdot \mathbf{y}}.$$

Further, $\mathcal{C}_{f,g}(\mathbf{u}) = \overline{\mathcal{C}_{g,f}(\mathbf{u})}$, for all $\mathbf{u} \in \mathbb{V}_n$, which implies that $\mathcal{C}_f(\mathbf{u})$ is always real.

(ii) If $f, g \in \mathcal{GB}_n^q$, then

$$\begin{aligned} \sum_{\mathbf{u} \in \mathbb{V}_n} \mathcal{C}_{f,g}(\mathbf{u})(-1)^{\mathbf{u} \cdot \mathbf{x}} &= 2^n \mathcal{H}_f(\mathbf{x}) \overline{\mathcal{H}_g(\mathbf{x})}, \\ \mathcal{C}_{f,g}(\mathbf{u}) &= \sum_{\mathbf{x} \in \mathbb{V}_n} \mathcal{H}_f(\mathbf{x}) \overline{\mathcal{H}_g(\mathbf{x})} (-1)^{\mathbf{u} \cdot \mathbf{x}}. \end{aligned}$$

(iii) Taking the particular case $f = g$ we obtain

$$\mathcal{C}_f(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbb{V}_n} |\mathcal{H}_f(\mathbf{x})|^2 (-1)^{\mathbf{u} \cdot \mathbf{x}}. \quad (5.1)$$

(iv) If $f \in \mathcal{GB}_n^q$, then f is a gbent function if and only if

$$\mathcal{C}_f(\mathbf{u}) = \begin{cases} 2^n & \text{if } \mathbf{u} = 0, \\ 0 & \text{if } \mathbf{u} \neq 0. \end{cases}$$

(v) Moreover, the (generalized) Parseval's identity holds

$$\sum_{\mathbf{x} \in \mathbb{V}_n} |\mathcal{H}_f(\mathbf{x})|^2 = 2^n. \quad (5.2)$$

Let $\zeta = e^{2\pi i/q}$ be the q -primitive root of unity, and $f : \mathbb{V}_n \rightarrow \mathbb{Z}_q$ as in (2.1). It turns out that the generalized Walsh–Hadamard spectrum of f can be described (albeit, in a complicated manner) in terms of the Walsh–Hadamard spectrum of its Boolean components a_i [44].

Theorem 5.5. [44] *The Walsh–Hadamard transform of $f : \mathbb{V}_n \rightarrow \mathbb{Z}_q$, $2^{h-1} < q \leq 2^h$, where $f(\mathbf{x}) = \sum_{i=0}^{h-1} a_i(\mathbf{x})2^i$, $a_i \in \mathcal{B}_n$ is given by*

$$\mathcal{H}_f(\mathbf{u}) = 2^{-h} \sum_{I \subseteq \{0, \dots, h-1\}} \zeta^{\sum_{i \in I} 2^i} \sum_{J \subseteq I, K \subseteq \bar{I}} (-1)^{|J|} W_{\sum_{\ell \in J \cup K} a_\ell(\mathbf{x})}(\mathbf{u}).$$

Proof. For brevity, we use the notations $\zeta_i := \zeta^{2^i}$. It is easy to see that, for $s \in \mathbb{Z}_2$, we have

$$z^s = \frac{1 + (-1)^s}{2} + \frac{1 - (-1)^s}{2} z, \quad (5.3)$$

and so, we have the identities $\zeta_i^{a_i(\mathbf{x})} = \frac{1}{2} (A_i + A'_i \zeta_i)$, where $A_i = 1 + (-1)^{a_i(\mathbf{x})}$, $A'_i = 1 - (-1)^{a_i(\mathbf{x})}$, and the complement $\bar{I} := \{0, 1, \dots, h-1\} \setminus I$, for some subset I of $\{0, 1, \dots, h-1\}$. The Walsh–Hadamard coefficients of f are

$$\begin{aligned} 2^{n/2} \mathcal{H}_f(\mathbf{u}) &= \sum_{\mathbf{x}} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{u} \cdot \mathbf{x}} = \sum_{\mathbf{x}} \zeta^{\sum_{i=0}^{h-1} a_i(\mathbf{x}) 2^i} (-1)^{\mathbf{u} \cdot \mathbf{x}} \\ &= \sum_{\mathbf{x}} (-1)^{\mathbf{u} \cdot \mathbf{x}} \prod_{i=0}^{h-1} \left(\zeta^{2^i} \right)^{a_i(\mathbf{x})} \\ &= \sum_{\mathbf{x}} (-1)^{\mathbf{u} \cdot \mathbf{x}} \prod_{i=0}^{h-1} \frac{1}{2} \left(1 + (-1)^{a_i(\mathbf{x})} + (1 - (-1)^{a_i(\mathbf{x})}) \zeta_i \right) \\ &= 2^{-h} \sum_{\mathbf{x}} (-1)^{\mathbf{u} \cdot \mathbf{x}} \sum_{I \subseteq \{0, \dots, h-1\}} \prod_{i \in I, j \in \bar{I}} \zeta_i A'_i A_j \\ &= 2^{-h} \sum_{\mathbf{x}} (-1)^{\mathbf{u} \cdot \mathbf{x}} \sum_{I \subseteq \{0, \dots, h-1\}} \zeta^{\sum_{i \in I} 2^i} \prod_{i \in I, j \in \bar{I}} A'_i A_j \\ &= 2^{-h} \sum_{\mathbf{x}} (-1)^{\mathbf{u} \cdot \mathbf{x}} \sum_{I \subseteq \{0, \dots, h-1\}} \zeta^{\sum_{i \in I} 2^i} \sum_{J \subseteq I, K \subseteq \bar{I}} (-1)^{|J|} (-1)^{\sum_{j \in J} a_j(\mathbf{x}) \oplus \sum_{k \in K} a_k(\mathbf{x})} \\ &= 2^{-h} \sum_{I \subseteq \{0, \dots, h-1\}} \zeta^{\sum_{i \in I} 2^i} \sum_{J \subseteq I, K \subseteq \bar{I}} (-1)^{|J|} \sum_{\mathbf{x}} (-1)^{\mathbf{u} \cdot \mathbf{x}} (-1)^{\sum_{\ell \in J \cup K} a_\ell(\mathbf{x})}, \end{aligned}$$

and so, we obtain our result. ■

5.3 Construction of Generalized Bent Functions in \mathcal{GB}_n^8

The material presented in this section was taken directly from the paper *Bent and generalized bent Boolean functions* [44] and appears in its original published form.

Theorem 5.6. [44] *If $f : \mathbb{V}_{n+2} \rightarrow \mathbb{Z}_8$ (n even) is given by*

$$f(\mathbf{x}, y, z) = 4c(\mathbf{x}) + (4a(\mathbf{x}) + 2c(\mathbf{x}) + 1)y + (4b(\mathbf{x}) + 2c(\mathbf{x}) + 1)z - 2yz,$$

where $a, b, c \in \mathcal{B}_n$ such that all $a, b, c, a \oplus c, b \oplus c$ and $a \oplus b$ are bent satisfying

$$W_a(\mathbf{x})W_b(\mathbf{x}) + W_{a \oplus c}(\mathbf{x})W_{b \oplus c}(\mathbf{x}) = -2W_{a \oplus b}(\mathbf{x})W_c(\mathbf{x}), \text{ for all } \mathbf{x} \in \mathbb{V}_n, \quad (5.4)$$

then f is gbent in \mathcal{GB}_{n+2}^8 .

Proof. We compute the Walsh–Hadamard coefficients (using that $\zeta = \frac{1}{\sqrt{2}}(1 + \iota)$ and $\zeta^2 = \iota$)

$$\begin{aligned} 2^{(n+2)/2} \mathcal{H}_f(\mathbf{u}, v, w) &= \sum_{(\mathbf{x}, y, z) \in \mathbb{V}_{n+2}} \zeta^{f(\mathbf{x}, y, z)} (-1)^{\mathbf{u} \cdot \mathbf{x} \oplus vy \oplus wz} \\ &= \sum_{\mathbf{x} \in \mathbb{V}_n} \zeta^{4c(\mathbf{x})} (-1)^{\mathbf{u} \cdot \mathbf{x}} \sum_{(y, z) \in \mathbb{V}_2} \zeta^{(4a(\mathbf{x}) + 2c(\mathbf{x}) + 1)y + (4b(\mathbf{x}) + 2c(\mathbf{x}) + 1)z - 2yz} (-1)^{vy \oplus wz} \\ &= \sum_{\mathbf{x} \in \mathbb{V}_n} (-1)^{c(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x}} \left(1 + (-1)^v (-1)^{a(\mathbf{x})} \iota^{c(\mathbf{x})} \zeta + (-1)^w (-1)^{b(\mathbf{x})} \iota^{c(\mathbf{x})} \zeta \right. \\ &\quad \left. + (-1)^{a(\mathbf{x}) \oplus b(\mathbf{x}) \oplus c(\mathbf{x}) \oplus v \oplus w} \right). \end{aligned}$$

Applying equation (5.3) with $(z, s) = (\iota, c(\mathbf{x}))$, that is, $\iota^{c(\mathbf{x})} = \frac{1 + (-1)^{c(\mathbf{x})}}{2} + \frac{1 - (-1)^{c(\mathbf{x})}}{2} \iota$, we obtain

$$\begin{aligned} 2\mathcal{H}_f(\mathbf{u}, v, w) &= W_c(\mathbf{u}) + \frac{(-1)^v \zeta}{2} (W_{a \oplus c}(\mathbf{u}) + W_a(\mathbf{u}) + \iota W_{a \oplus c}(\mathbf{u}) - \iota W_a(\mathbf{u})) \\ &\quad + \frac{(-1)^w \zeta}{2} (W_{b \oplus c}(\mathbf{u}) + W_b(\mathbf{u}) + \iota W_{b \oplus c}(\mathbf{u}) - \iota W_b(\mathbf{u})) + (-1)^{v \oplus w} W_{a \oplus b}(\mathbf{u}) \\ &= W_c(\mathbf{u}) + \frac{(-1)^v}{\sqrt{2}} (W_a(\mathbf{u}) + \iota W_{a \oplus c}(\mathbf{u})) \\ &\quad + \frac{(-1)^w}{\sqrt{2}} (W_b(\mathbf{u}) + \iota W_{b \oplus c}(\mathbf{u})) + (-1)^{v \oplus w} W_{a \oplus b}(\mathbf{u}). \end{aligned}$$

Therefore, the real and the imaginary parts of $\mathcal{H}_f(\mathbf{u}, v, w)$ are

$$\begin{aligned} \operatorname{Re}(\mathcal{H}_f(\mathbf{u}, v, w)) &= W_c(\mathbf{u}) + (-1)^{v \oplus w} W_{a \oplus b}(\mathbf{u}) + \frac{(-1)^v W_a(\mathbf{u}) + (-1)^w W_b(\mathbf{u})}{\sqrt{2}}, \\ \operatorname{Im}(\mathcal{H}_f(\mathbf{u}, v, w)) &= \frac{(-1)^v W_{a \oplus c}(\mathbf{u}) + (-1)^w W_{b \oplus c}(\mathbf{u})}{\sqrt{2}}. \end{aligned}$$

and so,

$$\begin{aligned} 4|\mathcal{H}_f(\mathbf{u}, v, w)|^2 &= \frac{1}{2} (W_a(\mathbf{u})^2 + W_b(\mathbf{u})^2 + W_{a \oplus c}(\mathbf{u})^2 + W_{b \oplus c}(\mathbf{u})^2 + 2W_c(\mathbf{u})^2 + 2W_{a \oplus b}(\mathbf{u})^2) \\ &\quad + (-1)^{v+w} (W_a(\mathbf{u})W_b(\mathbf{u}) + W_{a \oplus c}(\mathbf{u})W_{b \oplus c}(\mathbf{u}) + 2W_c(\mathbf{u})W_{a \oplus b}(\mathbf{u})) \\ &\quad + \sqrt{2} ((-1)^v (W_a(\mathbf{u})W_c(\mathbf{u}) + W_b(\mathbf{u})W_{a \oplus b}(\mathbf{u})) + (-1)^w (W_b(\mathbf{u})W_c(\mathbf{u}) + W_a(\mathbf{u})W_{a \oplus b}(\mathbf{u}))) \end{aligned} \quad (5.5)$$

Since $a, b, c, a \oplus c, b \oplus c, a \oplus b$ are all bent then $|W_a(\mathbf{u})| = |W_b(\mathbf{u})| = |W_c(\mathbf{u})| = |W_{a \oplus b}(\mathbf{u})| = |W_{a \oplus c}(\mathbf{u})| = |W_{b \oplus c}(\mathbf{u})| = 1$. Further, from the imposed conditions on these functions' Walsh–Hadamard coefficients, we see that $W_a(\mathbf{u})W_b(\mathbf{u}) + W_{a \oplus c}(\mathbf{u})W_{b \oplus c}(\mathbf{u}) + 2W_c(\mathbf{u})W_{a \oplus b}(\mathbf{u}) = 0$, and also $W_a(\mathbf{u})W_c(\mathbf{u}) + W_b(\mathbf{u})W_{a \oplus b}(\mathbf{u}) = 0$, $W_b(\mathbf{u})W_c(\mathbf{u}) + W_a(\mathbf{u})W_{a \oplus b}(\mathbf{u}) = 0$ (that is because if $W_a(\mathbf{u})$ and $W_b(\mathbf{u})$ have the same sign, then $W_c(\mathbf{u}), W_{a \oplus b}$ have opposite signs; further, $W_a(\mathbf{u})$ and $W_b(\mathbf{u})$ have opposite signs, then $W_c(\mathbf{u}), W_{a \oplus b}$ have the same sign). Using these equations, we get that $4|\mathcal{H}_f(\mathbf{u}, v, w)|^2 = 4$, and so, f is gbent [44].

■

5.4 Necessary Conditions for Generalized Bent Functions

The material presented in this section was taken directly from the paper *Generalized bent functions and their Gray images* [28] and appears in its original published form.

Theorem 5.7. [28] *All gbent functions $f \in \mathcal{GB}_n^{2^k}$ are regular, except for n odd and $k = 2$, in which case we have $\mathcal{H}_f^4(\mathbf{u}) = 2^{\frac{n-1}{2}}(\pm 1 \pm i)$.*

From the definition of a Boolean bent function via the Walsh-Hadamard transform we immediately obtain the following equivalent definition, where we denote the support of a Boolean function f by $\text{supp}(f) := \{\mathbf{x} \in \mathbb{V}_n : f(\mathbf{x}) = 1\}$: A Boolean function $f : \mathbb{V}_n \rightarrow \mathbb{F}_2$ is bent if and only if for every $\mathbf{u} \in \mathbb{V}_n$ the function $f_{\mathbf{u}}(\mathbf{x}) := f(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x}$ satisfies $|\text{supp}(f_{\mathbf{u}})| = 2^{n-1} \pm 2^{n/2}$. Our next target is to show an analog description for gbent functions. We use the following lemma [28].

Lemma 5.8. [28] *Let $q = 2^k$, $k > 1$, $\zeta = e^{2\pi i/q}$. If $\rho_l \in \mathbb{Q}$, $0 \leq l \leq q-1$ and $\sum_{l=0}^{q-1} \rho_l \zeta^l = r$ is rational, then $\rho_j = \rho_{2^{k-1}+j}$, for $1 \leq j \leq 2^{k-1}-1$ [28].*

Proof. Since $\zeta^{2^{k-1}+l} = -\zeta^l$ for $0 \leq l \leq 2^{k-1}-1$, we can write every element z of the cyclotomic field $\mathbb{Q}(\zeta)$ as

$$z = \sum_{l=0}^{2^{k-1}-1} \lambda_l \zeta^l, \lambda_l \in \mathbb{Q}, 0 \leq l \leq 2^{k-1}-1.$$

As $[\mathbb{Q}(\zeta) : \mathbb{Q}] = \varphi(q) = 2^{k-1}$ (φ is Euler's totient function), the set $\{1, \zeta, \dots, \zeta^{2^{k-1}-1}\}$ is a basis of $\mathbb{Q}(\zeta)$. Since

$$0 = \sum_{l=0}^{q-1} \rho_l \zeta^l - r = (\rho_0 - \rho_{2^{k-1}} - r) + \sum_{l=1}^{2^{k-1}-1} (\rho_l - \rho_{2^{k-1}+l}) \zeta^l.$$

the assertion of the lemma follows. ■

Proposition 5.9. [28] *Let $n = 2m$ be even, and for a function $f : \mathbb{V}_n \rightarrow \mathbb{Z}_{2^k}$ and $\mathbf{u} \in \mathbb{V}_n$, let $f_{\mathbf{u}}(\mathbf{x}) = f(\mathbf{x}) + 2^{k-1}(\mathbf{u} \cdot \mathbf{x})$, and let $b_j^{(\mathbf{u})} = |\{\mathbf{x} \in \mathbb{V}_n : f_{\mathbf{u}}(\mathbf{x}) = j\}|$, $0 \leq j \leq 2^k-1$. Then f is gbent if and only if for all $\mathbf{u} \in \mathbb{V}_n$ there exists an integer $\rho_{\mathbf{u}}$, $0 \leq \rho_{\mathbf{u}} \leq 2^{k-1}-1$ such that*

$$b_{2^{k-1}+\rho_{\mathbf{u}}}^{(\mathbf{u})} = b_{\rho_{\mathbf{u}}}^{(\mathbf{u})} \pm 2^m \text{ and } b_{2^{k-1}+j}^{(\mathbf{u})} = b_j^{(\mathbf{u})}, \text{ for } 0 \leq j \leq 2^{k-1}-1, j \neq \rho_{\mathbf{u}}.$$

Proof. First suppose that f is gbent. Then by Theorem 5.7, f is a regular gbent

function. Hence

$$\mathcal{H}_f(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbb{V}_n} \zeta^{f(\mathbf{x})} (-1)^{\mathbf{u} \cdot \mathbf{x}} = \sum_{\mathbf{x} \in \mathbb{V}_n} \zeta^{f(\mathbf{x}) + 2^{k-1}(\mathbf{u} \cdot \mathbf{x})} = \mathcal{H}_{f_{\mathbf{u}}}(0) = \sum_{j=0}^{2^k-1} b_j^{(\mathbf{u})} \zeta^j = 2^m \zeta^r$$

for some $0 \leq r \leq 2^k - 1$. With $\rho_{\mathbf{u}} = r$ if $0 \leq r \leq 2^{k-1} - 1$, and $\rho_{\mathbf{u}} = r - 2^{k-1}$ otherwise, the claim follows from Lemma 5.8.

The converse statement is verified in a straightforward manner [28]. ■

We now can present connections between gbent functions and their components for the general case of gbent functions in $\mathcal{GB}_n^{2^k}$, $k > 1$. This generalizes the corresponding results for $k = 2$ and $k = 3$ in [42] and in [44].

Theorem 5.10. [28] *Let n be even, and let $f(\mathbf{x})$ be a gbent function in $\mathcal{GB}_n^{2^k}$, $k > 1$, (uniquely) given as*

$$f(\mathbf{x}) = a_1(\mathbf{x}) + 2a_2(\mathbf{x}) + \cdots + 2^{k-2}a_{k-1}(\mathbf{x}) + 2^{k-1}a_k(\mathbf{x}),$$

$a_i \in \mathcal{B}_n$, $1 \leq i \leq k$. Then all Boolean functions of the form

$$g_{\mathbf{c}}(\mathbf{x}) = c_1a_1(\mathbf{x}) \oplus c_2a_2(\mathbf{x}) \oplus \cdots \oplus c_{k-1}a_{k-1}(\mathbf{x}) \oplus a_k(\mathbf{x}),$$

$\mathbf{c} = (c_1, c_2, \dots, c_{k-1}) \in \mathbb{F}_2^{n-1}$, are bent functions.

Proof. As in Proposition 5.9, for the gbent function f we denote by $f_{\mathbf{u}}$ the function $f_{\mathbf{u}}(\mathbf{x}) = a_1(\mathbf{x}) + \cdots + 2^{k-2}a_{k-1}(\mathbf{x}) + 2^{k-1}(a_k(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x})$ in $\mathcal{GB}_n^{2^k}$. Again, the integer $b_r^{(\mathbf{u})}$, $0 \leq r \leq 2^k - 1$, is defined as $b_r^{(\mathbf{u})} = |\{\mathbf{x} \in \mathbb{V}_n : f_{\mathbf{u}}(\mathbf{x}) = r\}|$. By Proposition 5.9, $b_{r+2^{k-1}}^{(\mathbf{u})} = b_r^{(\mathbf{u})}$ for all $0 \leq r \leq 2^{k-1} - 1$, except for one element $\rho_{\mathbf{u}} \in \{0, \dots, 2^{k-1} - 1\}$ depending on \mathbf{u} , for which $b_{\rho_{\mathbf{u}}+2^{k-1}}^{(\mathbf{u})} = b_{\rho_{\mathbf{u}}}^{(\mathbf{u})} \pm 2^{n/2}$.

Since it is somewhat easier to follow, we first show the bentness of $a_k(\mathbf{x}) = g_0(\mathbf{x})$. In the second step we show the general case. For $r \neq \rho_{\mathbf{u}}$, $0 \leq r \leq 2^{k-1} - 1$, consider all $\mathbf{x} \in \mathbb{V}_n$ for which $a_1(\mathbf{x}) + \cdots + 2^{k-2}a_{k-1}(\mathbf{x}) = r$. Since $b_{r+2^{k-1}}^{(\mathbf{u})} = b_r^{(\mathbf{u})}$, for exactly half of these \mathbf{x} we have $a_k(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x} = 0$ (note that the number of these \mathbf{x} must be even). Among all $\mathbf{x} \in \mathbb{V}_n$ for which $a_1(\mathbf{x}) +$

$\dots + 2^{k-2}a_{k-1}(\mathbf{x}) = \rho_{\mathbf{u}}$, there are $b_{\rho_{\mathbf{u}}}^{(\mathbf{u})}$ for which $a_k(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x} = 0$, and there are $b_{\rho_{\mathbf{u}}+2^{k-1}}^{(\mathbf{u})} = b_{\rho_{\mathbf{u}}}^{(\mathbf{u})} \pm 2^{n/2}$ for which $a_k(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x} = 1$. Hence for the Walsh-Hadamard transform of a_k we get

$$\mathcal{W}_{a_k}(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbb{V}_n} (-1)^{a_k(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x}} = \pm 2^{n/2},$$

which shows that a_k is bent.

To show that $g_{\mathbf{c}}$ is bent for every $\mathbf{c} \in \mathbb{F}_2^{k-1}$, we write $f_{\mathbf{u}}(\mathbf{x})$, $\mathbf{u} \in \mathbb{V}_n$, as

$$\begin{aligned} f_{\mathbf{u}}(\mathbf{x}) &= c_1 a_1(\mathbf{x}) + \dots + c_{k-1} 2^{k-2} a_{k-1}(\mathbf{x}) + \bar{c}_1 a_1(\mathbf{x}) + \dots + \bar{c}_{k-1} 2^{k-2} a_{k-1}(\mathbf{x}) \\ &\quad + 2^{k-1}(a_k(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x}) := h(\mathbf{x}) + \bar{h}(\mathbf{x}) + 2^{k-1}(a_k(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x}), \end{aligned}$$

where $\bar{c} = c \oplus 1$. Note that every $0 \leq r \leq 2^{k-1} - 1$ in the value set of $a_1(\mathbf{x}) + \dots + 2^{k-2}a_{k-2}(\mathbf{x})$ has then a unique representation as $h(\mathbf{x}) + \bar{h}(\mathbf{x})$. Consider \mathbf{x} for which $h(\mathbf{x}) + \bar{h}(\mathbf{x}) = r + s \neq \rho_{\mathbf{u}}$. Again from $b_{r+2^{k-1}}^{(\mathbf{u})} = b_r^{(\mathbf{u})}$ we infer that for half of those \mathbf{x} we have $a_k(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x} = 0$. Hence also

$$g_{\mathbf{c}}(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x} = c_1 a_1(\mathbf{x}) \oplus \dots \oplus c_{k-1} a_{k-1}(\mathbf{x}) \oplus a_k(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x} = 0$$

for exactly half of those \mathbf{x} . (Observe that $h(\mathbf{x}_1) = h(\mathbf{x}_2) = r$ implies $c_1 a_1(\mathbf{x}_1) \oplus \dots \oplus c_{k-1} a_{k-1}(\mathbf{x}_1) = c_1 a_1(\mathbf{x}_2) \oplus \dots \oplus c_{k-1} a_{k-1}(\mathbf{x}_2)$.) Similarly as above, among all $\mathbf{x} \in \mathbb{V}_n$ for which $h(\mathbf{x}) + \bar{h}(\mathbf{x}) = \rho_{\mathbf{u}}$, there are $b_{\rho_{\mathbf{u}}}^{(\mathbf{u})}$ for which $a_k(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x} = 0$, and there are $b_{\rho_{\mathbf{u}}+2^{k-1}}^{(\mathbf{u})} = b_{\rho_{\mathbf{u}}}^{(\mathbf{u})} \pm 2^{n/2}$ for which $a_k(\mathbf{x}) \oplus \mathbf{u} \cdot \mathbf{x} = 1$. From this we conclude that $|\{\mathbf{x} \in \mathbb{V}_n : h(\mathbf{x}) + \bar{h}(\mathbf{x}) = \rho_{\mathbf{u}} \text{ and } f_{\mathbf{u}}(\mathbf{x}) = 1\}| - |\{\mathbf{x} \in \mathbb{V}_n : h(\mathbf{x}) + \bar{h}(\mathbf{x}) = \rho_{\mathbf{u}} \text{ and } f_{\mathbf{u}}(\mathbf{x}) = 0\}| = \pm 2^{n/2}$. Therefore

$$\mathcal{W}_{g_{\mathbf{c}}}(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbb{V}_n} (-1)^{g_{\mathbf{c}}(\mathbf{x}) + \mathbf{u} \cdot \mathbf{x}} = \pm 2^{n/2},$$

and $g_{\mathbf{c}}$ is bent [28]. ■

Theorem 5.10, which assigns to a gbent function an affine space of bent functions, provides a necessary condition for a function $f \in \mathcal{GB}_n^{2^k}$ to be gbent. For $k > 2$ the condition is not sufficient [28].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:

Conclusion and Future Research

Set your course by the stars, not by the
lights of every passing ship.

Omar N. Bradley★★★★

6.1 Conclusion

In this dissertation we investigated generalized Boolean functions which were correlation immune, satisfied various avalanche features, and which were generalized bent. We presented several construction techniques for order 1 and higher correlation immune generalized Boolean functions, and also established new avalanche criteria for generalized Boolean functions. The goal of this research has been to increase our understanding of the inherent attributes of generalized Boolean functions so that we are capable of making prudent design choices when selecting these functions as components in future encryption schemes. Along the way we discovered several parallels between these functions and their Boolean counterparts, but oftentimes saw that things become more complicated when operating in a q -ary environment. In particular, we showed that while the Walsh-Hadamard transform is an outstanding tool for establishing whether or not Boolean functions satisfy certain cryptographic properties such as balance and correlation immunity, its utility is somewhat diminished in the more generalized setting. One area of concern which we attempted to address was the potential of adversaries carrying out what we termed was a “decomposition attack” whereby they perform a binary expansion of the q -ary functional outputs in an attempt to discover weaknesses in the underlying Boolean function components. We showed that correlation immune generalized Boolean functions will not succumb to such techniques, but that when it comes to avalanche criteria, more care must be taken. One family of generalized Boolean functions which we believe shows particular promise are those that satisfy the uniform avalanche criterion. These functions are both probabilistic SAC as well as 1-resilient (order 1 correlation immune and balanced). Moreover, their con-

stituent Boolean functions are guaranteed to also be resilient and *SAC*, thus making these functions resistant to decomposition attacks targeting these properties. Like many things in cryptography, trade-off and compromises abound. While generalized Boolean functions will most likely find their rightful place in certain applications, they will equally likely prove unsuitable for others.

6.2 Future Research

We briefly investigated linear structures and directional derivatives of *UAC* compliant generalized Boolean function, demonstrating the utility in ensuring that equal probabilities exist among the unit vectors when the generalized Boolean function's derivatives equals zero. It would be interesting in future research to further investigate linear structures of generalized Boolean functions, including the Meier and Staffelbach approach of perfect nonlinear generalized Boolean functions [30], as well as a notion of almost perfect nonlinear (*APN*) for generalized Boolean functions. We would also like to find a proof of Conjecture 2.26 and thus prove that there can be no symmetric and balanced generalized Boolean functions.

APPENDIX A:

Table of Nontrivial Binomial Bisections

The following table of nontrivial bisection solutions is a copy of the table which appears in the coauthored paper *Bisecting binomial coefficients* which was published in the journal *Discrete Applied Mathematics* [27].

The table contains the complete set of nontrivial bisection solution vectors for $1 \leq n \leq 50$. In the interest of saving space, we only list the highest lexicographically occurring solutions. Any additional solutions which a listed solution may yield, can be generated in the following manner: If a pair of bits are equidistant from the center of the given vector and differ, they may both be negated to produce a new solution. Additionally, any solution vector can also be reversed and negated in its entirety to produce yet another solution [27].

Table A.1: Nontrivial binomial bisections

n	number of solutions	nontrivial solution vectors
8	4	100110001
13	16	11110011001000
14	4	101001101000101
	8	101011100100101
20	4	101010011010100010101
24	32	1000110111011000100010001
	16	1011001111010100101000101
26	4	101010100110101010001010101
29	2048	111111110111011000110010000000
31	512	11110110011111100010101000001000
	128	11110110010110011001100000001000
32	4	101010101001101010101000101010101
33	16384	111111111111110011010010000000000000
34	64	10101001110110111010000000110010101
	32	10101001110111101010010000110010101
	16	10101001111100111010000110110010101
	8	10101001111101101010010110110010101
	8	10101010101011011010001010101010101
35	8	101010101010100111001001010101010101
	16	101010101011100111001000110101010101
38	4	1010101010100110101010101000101010101
	32	101111110010111110100011100010011011101
41	2048	111111011110101001111000100100001110100000
	4096	111111011110111001111000100010001110100000
	8192	111111111111001010111001000100100010100000
	16384	111111111111011010111001000010100010100000
44	4	1010101010101001101010101010100010101010101
	128	101011111000111111110110000011011000110110101
47	1048576	111111111111110100111111000001000000100000000000
48	4096	1011001111011011010111010101000000000001000000101
50	4	1010101010101010011010101010101010001010101010101

APPENDIX B:

Binomial Bisection Program

The following parallel computer program written in Julia was created by the dissertation author to exhaustively search for all nontrivial bisection solutions. The code was run on the Hamming high performance computer (HPC) at the Naval Postgraduate School, and found all nontrivial binomial bisections for $n \leq 51$ (see Appendix A.1). In addition to the cited paper, these research results also contributed to the integer sequence, for $37 \leq n \leq 51$, of the total number of binomial bisection (trivial and nontrivial) which appears as A200147 in the Online Encyclopedia of Integer Sequences.

```
using MPI

function bisect(n, q, p, r)
    # typealias BInt_t UInt8
    # typealias BInt_t UInt16
    # typealias BInt_t UInt32
    # typealias BInt_t UInt64
    # typealias BInt_t UInt128

    @assert BInt_t != UInt8    || n < 8
    @assert BInt_t != UInt16   || n < 16
    @assert BInt_t != UInt32   || n < 32
    @assert BInt_t != UInt64   || n < 64
    @assert BInt_t != UInt128  || n < 128

    comm = MPI.COMM_WORLD
    root = 0
    rank = MPI.Comm_rank(comm)
    size = MPI.Comm_size(comm)
    const procs = q
    const pgms = p
    const pgm_inst = r
```

```

const stride :: BInt_t = Int(2^(n+1)/(procs*pgms))

#Set target value
const bisect_sum = BInt_t(2)^(n-1)

#Set vector center
const center = div(n+1,2)

#Set lower Hamming weight boundry
if n <=4
    lwr_wt = 0
elseif 4 < n <= 6
    lwr_wt = 2
elseif 7 <= n <= 8
    lwr_wt = 3
elseif 8 < n <= 10
    lwr_wt = 4
elseif 10 < n <=12
    lwr_wt = 5
else
    m = Int(ceil(log(2,n)))
    lwr_wt = Int(ceil(log(2,m))) + m
end

#Set upper Hamming weight boundry
if n <=4
    upr_wt = n+1
else
    upr_wt = n-lwr_wt+1
end

#Create binary coefficients array
bin_coef = Array(BInt_t, n+1)
for j = 1:n+1

```

```

        @inbounds bin_coef[j] = binomial(big(n),j-1)
    end

#Initialize solution containers
    solns = []
    count = 0

    function gather(obj, root::Integer, comm::MPI.Comm)
        buf = Array{typeof(obj), MPI.Comm_size(comm)}
        if (MPI.Comm_rank(comm) != root)
            MPI.send(obj, root, 666, comm)
        else
            for r = 0:MPI.Comm_size(comm)-1
                if r != root
                    rmesg = MPI.recv(r, 666, comm)
                    buf[r+1] = rmesg[1]
                else
                    buf[r+1] = obj
                end
            end
        end
    end

    buf
end

#Test for symmetry
function sym_test(f)
    i = 1
    @inbounds while i <= center
        if f[i] != f[n+2-i]
            return 0
            break
        else
            i += 1
        end
    end
end

```

```

        end
        return 1
    end

#Eliminate vectors
function test(f)
    issym = true
    j=1
    while j <= center
        if f[j] == 1 && f[n+2-j] == 0
            return 0
        elseif issym && f[j] != f[n+2-j]
            issym = false
        end
        j += 1
    end
    if issym && f[1] == 1
        return 0
    end
    return 1
end

#Generate solution vectors
function gen_sol(f)
    if sym_test(f) == 1
        count += 2
    else
        j = 0
        for i = 1:center
            if f[i] != f[n+2-i]
                j += 1
            end
        end
        count += 2^j
    end
end

```



```

        end
    end

#Check candidate vectors
    f = zeros(BInt_t,n+1);
    BInt_zero = BInt_t(0)
    BInt_one = BInt_t(1)

    start::BInt_t = pgm_inst*(procs*stride) + (rank*stride) + 1
    stop::BInt_t = (start + stride) - 1

    for s = start : stop
        sum_f = BInt_zero
        for i = 1:n+1
            tmp = s & BInt_one
            s = s >> 1
            f[i] = tmp
            sum_f += tmp
            if sum_f >= upr_wt
                break
            end
        end
    end

    if lwr_wt < sum_f < upr_wt
        if test(f) == 1
            my_sum = BInt_t(0)
            for k = 1:n+1
                if f[k] == 1
                    my_sum += bin_coef[k]
                end
                if my_sum > bisect_sum
                    break
                end
            end
        end
    end
end

```

```

        if bisect_sum == my_sum
            println(round(Int,f))
            gen_sol(f)
        end
    end
end
end
end

gcount = gather(count, root, comm)
if(rank == root)
    num_sol = sum(gcount)
    println("N = ",n, " Section = ",pgm_inst, "; Number of Bisections = ",
        num_sol)

end
end
let
    MPI.Init()
    n = 51
    q = 64
    p = 64
    r = 53

    if (MPI.Comm_rank(MPI.COMM_WORLD) == 0)
        tic()

    end

    bisect(n, q, p, r)
    if (MPI.Comm_rank(MPI.COMM_WORLD) == 0)
        toc()

    end
    MPI.Finalize()
end
end

```

APPENDIX C:

Some Linear Orthogonal Arrays for Higher Order Correlation Immune Generalized Boolean Function Constructions

The following (incomplete) list of linear orthogonal arrays which are suitable for constructing higher order correlation immune generalized Boolean functions using the method outlined in Algorithm 4, have been compiled using data from Hedayat, Sloane and Stufken's book on orthogonal arrays [19] as well as the Sloan online database of orthogonal arrays [41].

OA(8,5,2,2):	OA(8,7,2,2):	OA(16,8,2,3):	OA(16,8,2,3):
00000	0000000	00000000	00000000
10011	1010101	01010101	00101110
01010	0110011	00110011	01010110
00101	1100110	01100110	01111000
11001	0001111	00001111	10011010
10110	1011010	01011010	10110100
01111	0111100	00111100	11001100
11100	1101001	01101001	11100010
		11111111	11111111
		10101010	11010001
		11001100	10101001
		10011001	10000111
		11110000	01100101
		10100101	01001011
		11000011	00110011
		10010110	00011101

OA(128,9,2,5):

000000000	010000010	110000001	100000011
111000000	101000010	001000001	011000011
100100000	110100010	010100001	000100011
011100000	001100010	101100001	111100011
010010000	000010010	100010001	110010011
101010000	111010010	011010001	001010011
110110000	100110010	000110001	010110011
001110000	011110010	111110001	101110011
110001000	100001010	000001001	010001011
001001000	011001010	111001001	101001011
010101000	000101010	100101001	110101011
101101000	111101010	011101001	001101011
100011000	110011010	010011001	000011011
011011000	001011010	101011001	111011011
000111000	010111010	110111001	100111011
111111000	101111010	001111001	011111011
100000100	110000110	010000101	000000111
011000100	001000110	101000101	111000111
000100100	010100110	110100101	100100111
111100100	101100110	001100101	011100111
110010100	100010110	000010101	010010111
001010100	011010110	111010101	101010111
010110100	000110110	100110101	110110111
101110100	111110110	011110101	001110111
010001100	000001110	100001101	110001111
101001100	111001110	011001101	001001111
110101100	100101110	000101101	010101111
001101100	011101110	111101101	101101111
000011100	010011110	110011101	100011111
111011100	101011110	001011101	011011111
100111100	110111110	010111101	000111111
011111100	001111110	101111101	111111111

OA(16,15,2,2):

0000000000000000
101010101010101
011001100110011
110011001100110
000111100001111
101101001011010
011110000111100
110100101101001
000000011111111
101010110101010
011001111001100
110011010011001
000111111110000
101101010100101
011110011000011
110100110010110

OA(16,15,2,2):

0000000000000000
101010101010101
011001100110011
110011001100110
000111100001111
101101001011010
011110000111100
110100101101001
000000011111111
101011010101001
011001111001100
110010110011010
000111111110000
101100110100110
011110011000011
110101010010101

OA(16,15,2,2):

0000000000000000
101010101010101
011001100110011
110011001100110
000111100001111
101101001011010
011110000111100
110100101101001
000000011111111
000111111110000
011010111001010
011101011000101
101011010101001
101100110100110
110001110011100
110110010010011

OA(16,15,2,2):

0000000000000000
 101010101010101
 011001100110011
 110011001100110
 000111100001111
 101101001011010
 011110000111100
 110100101101001
 000000011111111
 001011111101000
 010111011010001
 011100111000110
 100101110110100
 101110010100011
 110010110011010
 111001010001101

OA(16,15,2,2):

0000000000000000
 101010100001111
 011001100110011
 110011001010101
 000111100111100
 101101001100110
 011110001101001
 110100101011010
 000000011111111
 101010111110000
 011001111001100
 110011010101010
 000111111000011
 101101010011001
 011110010010110
 110100110100101

OA(32,16,2,3):

0000000000000000
 0101010101010101
 0011001100110011
 0110011001100110
 0000111100001111
 0101101001011010
 0011110000111100
 0110100101101001
 0000000011111111
 0101010110101010
 0011001111001100
 0110011010011001
 0000111111110000
 0101101010100101
 0011110011000011
 0110100110010110
 1111111111111111
 1010101010101010
 1100110011001100
 1001100110011001
 1111000011110000
 1010010110100101
 1100001111000011
 1001011010010110
 1111111100000000
 1010101001010101
 1100110000110011
 1001100101100110
 1111000000001111
 1010010101011010
 1100001100111100
 1001011001101001

OA(32,16,2,3):

0000000000000000
0101010101010101
0011001100110011
0110011001100110
0000111100001111
0101101001011010
0011110000111100
0110100101101001
0000000011111111
0101011010101001
0011001111001100
0110010110011010
0000111111110000
0101100110100110
0011110011000011
0110101010010101
1111111111111111
1010101010101010
1100110011001100
1001100110011001
1111000011110000
1010010110100101
1100001111000011
1001011010010110
1111111100000000
1010100101010110
1100110000110011
1001101001100101
1111000000001111
1010011001011001
1100001100111100
1001010101101010

OA(32,16,2,3):

0000000000000000
0101010101010101
0011001100110011
0110011001100110
0000111100001111
0101101001011010
0011110000111100
0110100101101001
0000000011111111
0000111111110000
0011010111001010
0011101011000101
0101011010101001
0101100110100110
0110001110011100
0110110010010011
1111111111111111
1010101010101010
1100110011001100
1001100110011001
1111000011110000
1010010110100101
1100001111000011
1001011010010110
1111111100000000
1111000000001111
1100101000110101
1100010100111010
1010100101010110
1010011001011001
1001110001100011
1001001101101100

OA(32,16,2,3):

0000000000000000
0101010101010101
0011001100110011
0110011001100110
0000111100001111
0101101001011010
0011110000111100
0110100101101001
0000000011111111
0001011111101000
0010111011010001
0011100111000110
0100101110110100
0101110010100011
0110010110011010
0111001010001101
1111111111111111
1010101010101010
1100110011001100
1001100110011001
1111000011110000
1010010110100101
1100001111000011
1001011010010110
1111111100000000
1110100000010111
1101000100101110
1100011000111001
1011010001001011
1010001101011100
1001101001100101
1000110101110010

OA(32,16,2,3):

0000000000000000
0101010100001111
0011001100110011
0110011001010101
0000111100111100
0101101001100110
0011110001101001
0110100101011010
0000000011111111
0101010111110000
0011001111001100
0110011010101010
0000111111000011
0101101010011001
0011110010010110
0110100110100101
1111111111111111
1010101011110000
1100110011001100
1001100110101010
1111000011000011
1010010110011001
1100001110010110
1001011010100101
1111111100000000
1010101000001111
1100110000110011
1001100101010101
1111000000111100
1010010101100110
1100001101101001
1001011001011010

OA(64,32,2,3):

00000000000000000000000000000000
01010101010101010101010101010101
00110011001100110011001100110011
01100110011001100110011001100110
00001111000011110000111100001111
01011010010110100101101001011010
00111100001111000011110000111100
01101001011010010110100101101001
00000000111111110000000011111111
01010101101010100101010110101010
00110011110011000011001111001100
01100110100110010110011010011001
00001111111100000000111111110000
01011010101001010101101010100101
00111100110000110011110011000011
01101001100101100110100110010110
00000000000000001111111111111111
01010101010101010110101010101010
00110011001100111100110011001100
01100110011001101001100110011001
00001111000011111111000011110000
01011010010110101010010110100101
00111100001111001100001111000011
01101001011010011001011010010110
00000000111111111111111100000000
01010101101010101010101001010101
00110011110011001100110000110011
01100110100110011001100101100110
00001111111100001111000000001111
010110101010010110100101011010
00111100110000111100001100111100
01101001100101101001011001101001

11111111111111111111111111111111
10101010101010101010101010101010
11001100110011001100110011001100
10011001100110011001100110011001
11110000111100001111000011110000
10100101101001011010010110100101
11000011110000111100001111000011
10010110100101101001011010010110
11111111000000001111111100000000
10101010010101011010101001010101
11001100001100111100110000110011
10011001011001101001100101100110
11110000000011111111000000001111
10100101010110101010010101011010
11000011001111001100001100111100
10010110011010011001011001101001
11111111111111110000000000000000
10101010101010100101010101010101
11001100110011000011001100110011
10011001100110010110011001100110
11110000111100000000111100001111
10100101101001010101101001011010
11000011110000110011110000111100
10010110100101100110100101101001
11111111000000000000000011111111
10101010010101010101010110101010
11001100001100110011001111001100
10011001011001100110011010011001
11110000000011110000111111110000
10100101010110100101101010100101
11000011001111000011110011000011
10010110011010010110100110010110

OA(64,32,2,3):

00000000000000000000000000000000
01010101010101010101010101010101
00110011001100110011001100110011
01100110011001100110011001100110
00001111000011110000111100001111
01011010010110100101101001011010
00111100001111000011110000111100
01101001011010010110100101101001
00000000111111110000000011111111
01010110101010010101011010101001
00110011110011000011001111001100
01100101100110100110010110011010
00001111111100000000111111110000
01011001101001100101100110100110
00111100110000110011110011000011
01101010100101010110101010010101
00000000000000001111111111111111
01010101010101011010101010101010
00110011001100111100110011001100
01100110011001101001100110011001
00001111000011111111000011110000
01011010010110101010010110100101
00111100001111001100001111000011
01101001011010011001011010010110
00000000111111111111111100000000
01010110101010011010100101010110
00110011110011001100110000110011
01100101100110101001101001100101
00001111111100001111000000001111
01011001101001101010011001011001
00111100110000111100001100111100
01101010100101011001010101010101

11111111111111111111111111111111
10101010101010101010101010101010
11001100110011001100110011001100
10011001100110011001100110011001
11110000111100001111000011110000
10100101101001011010010110100101
11000011110000111100001111000011
10010110100101101001011010010110
11111111000000001111111100000000
10101001010101101010100101010110
11001100001100111100110000110011
10011010011001011001101001100101
11110000000011111111000000001111
10100110010110011010011001011001
11000011001111001100001100111100
10010101011010101001010101101010
11111111111111110000000000000000
10101010101010100101010101010101
11001100110011000011001100110011
10011001100110010110011001100110
11110000111100000000111100001111
10100101101001010101101001011010
11000011110000110011110000111100
10010110100101100110100101101001
11111111000000000000000011111111
10101001010101100101011010101001
11001100001100110011001111001100
10011010011001010110010110011010
11110000000011110000111111110000
10100110010110010101100110100110
11000011001111000011110011000011
10010101011010100110101010010101

OA(64,32,2,3):

00000000000000000000000000000000
01010101010101010101010101010101
00110011001100110011001100110011
01100110011001100110011001100110
00001111000011110000111100001111
01011010010110100101101001011010
00111100001111000011110000111100
01101001011010010110100101101001
00000000111111110000000011111111
00001111111100000000111111110000
00110101110010100011010111001010
00111010110001010011101011000101
01010110101010010101011010101001
01011001101001100101100110100110
01100011100111000110001110011100
01101100100100110110110010010011
00000000000000001111111111111111
01010101010101011010101010101010
00110011001100111100110011001100
01100110011001101001100110011001
00001111000011111111000011110000
01011010010110101010010110100101
00111100001111001100001111000011
01101001011010011001011010010110
00000000111111111111111100000000
00001111111100001111000000001111
00110101110010101100101000110101
00111010110001011100010100111010
010101101010100110101001010110
01011001101001101010011001011001
01100011100111001001110001100011
01101100100100111001001101101100

11111111111111111111111111111111
10101010101010101010101010101010
11001100110011001100110011001100
10011001100110011001100110011001
11110000111100001111000011110000
10100101101001011010010110100101
11000011110000111100001111000011
10010110100101101001011010010110
11111111000000001111111100000000
11110000000011111111000000001111
11001010001101011100101000110101
11000101001110101100010100111010
10101001010101101010100101010110
10100110010110011010011001011001
10011100011000111001110001100011
10010011011011001001001101101100
11111111111111110000000000000000
10101010101010100101010101010101
11001100110011000011001100110011
10011001100110010110011001100110
11110000111100000000111100001111
10100101101001010101101001011010
11000011110000110011110000111100
10010110100101100110100101101001
11111111000000000000000011111111
11110000000011110000111111110000
11001010001101010011010111001010
11000101001110100011101011000101
10101001010101100101011010101001
10100110010110010101100110100110
10011100011000110110001110011100
10010011011011000110110010010011

OA(64,32,2,3):

00000000000000000000000000000000
01010101010101010101010101010101
00110011001100110011001100110011
01100110011001100110011001100110
00001111000011110000111100001111
01011010010110100101101001011010
00111100001111000011110000111100
01101001011010010110100101101001
00000000111111110000000011111111
00010111111010000001011111101000
00101110110100010010111011010001
00111001110001100011100111000110
01001011101101000100101110110100
01011100101000110101110010100011
01100101100110100110010110011010
01110010100011010111001010001101
00000000000000001111111111111111
01010101010101011010101010101010
00110011001100111100110011001100
01100110011001101001100110011001
00001111000011111111000011110000
01011010010110101010010110100101
00111100001111001100001111000011
01101001011010011001011010010110
00000000111111111111111100000000
00010111111010001110100000010111
00101110110100011101000100101110
00111001110001101100011000111001
01001011101101001011010001001011
01011100101000111010001101011100
01100101100110101001101001100101
01110010100011011000110101110010

11111111111111111111111111111111
10101010101010101010101010101010
11001100110011001100110011001100
10011001100110011001100110011001
11110000111100001111000011110000
10100101101001011010010110100101
11000011110000111100001111000011
10010110100101101001011010010110
11111111100000000111111110000000
11101000000101111110100000010111
11010001001011101101000100101110
11000110001110011100011000111001
10110100010010111011010001001011
10100011010111001010001101011100
10011010011001011001101001100101
10001101011100101000110101110010
11111111111111111000000000000000
10101010101010100101010101010101
11001100110011000011001100110011
10011001100110010110011001100110
11110000111100000000111100001111
10100101101001010101101001011010
11000011110000110011110000111100
10010110100101100110100101101001
11111111100000000000000001111111
11101000000101110001011111101000
11010001001011100010111011010001
11000110001110010011100111000110
10110100010010110100101110110100
10100011010111000101110010100011
10011010011001010110010110011010
10001101011100100111001010001101

LIST OF REFERENCES

- [1] D.K. Biss, “A lower bound on the number of functions satisfying the strict avalanche criterion,” *Discrete Math.*, vol. 185, pp. 29–39, 1998.
- [2] R.C. Bose, “On some connections between the design of experiments and information theory,” *Bull. Internat. Statist. Inst.*, vol. 38, no. 4, pp. 257–271.
- [3] P. Camion, C. Carlet, P. Charpin, and N. Sendrier, “On correlation-immune functions,” in *Adv. in Crypt. – Crypto ’91*, ser. LNCS, vol. 576, Berlin, Springer, 1992, pp. 86–100.
- [4] C. Carlet, “ \mathbb{Z}_2^k -linear Codes,” *IEEE Trans. Inf. Theory*, vol. 44, no. 4, pp. 1543–1547, 1998.
- [5] C. Carlet and P. Gaborit, “Hyper-bent functions and cyclic codes,” *J. Combin. Theory*, ser. A, vol. 113, pp. 446–482, 2006.
- [6] C. Carlet and P. Guillot, “A characterization of binary bent functions,” *J. Combin. Theory*, ser. A, vol. 76, no. 2, pp. 328–335, 1996.
- [7] C. Carlet, “Boolean Functions for Cryptography and Error Correcting Codes,” in *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, Yves Crama and Peter L. Hammer eds., New York, NY, Cambridge University Press, 2010, pp. 257–397.
- [8] C. Carlet, “Vectorial Boolean Functions for Cryptography,” in *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, Yves Crama and Peter L. Hammer eds., New York, NY, Cambridge University Press, 2010, pp. 398–469.
- [9] G. Chartrand and P. Zhang, *Introduction to Graph Theory*, New York, NY, McGraw-Hill, 2005, p. 25.
- [10] T.W. Cusick and Y. Li, “ k -th order symmetric SAC boolean functions and bisecting binomial coefficients,” *Discrete Appl. Math.*, vol. 149, pp. 73–86, 2005.
- [11] T.W. Cusick and P. Stănică, *Cryptographic Boolean Functions and Applications*, 2nd ed., San Diego, CA, Academic Press, 2017.
- [12] P. Delsarte, “An algebraic approach to the association schemes of coding theory,” *Philips Res. Reports Suppl.*, no. 10, p. 10, 1973.
- [13] J.F. Dillon, “Elementary Hadamard difference sets,” Ph.D. dissertation, Dept. of Mathematics, University of Maryland, College Park, MD, 1974.
- [14] H. Feistel, “Cryptography and Computer Privacy,” *Scientific American*, vol. 228, no. 5, p. 15, 1973.
- [15] E. Filiol and C. Fontain, “Highly nonlinear balanced Boolean functions with good correlation immunity,” in *Adv. in Crypt. – Eurocrypt ’98*, ser. LNCS, vol. 403, Berlin, Springer, 1998, pp. 475–488.
- [16] K. Gopalakrishnan, D.G. Hoffman, and D.R. Stinson, “A note on a conjecture concerning symmetric resilient functions,” *Inform. Proc. Lett.*, vol. 47, pp. 139–143, 1993.

- [17] S. Gangopadhyay, E. Pasalic, and P. Stănică, “A note on generalized bent criteria for Boolean functions,” *IEEE Trans. Inform. Theory*, vol. 59, no. 5, pp. 3233–3236, 2013.
- [18] J. von zur Gathen and J. Roche, “Polynomials with two values,” *Combinatorica*, vol. 17, pp. 345–362, 1997.
- [19] A.S. Hedayat, N.J.A. Sloane, and J. Stufken, *Orthogonal Arrays: theory and applications*, New York, NY, Springer, 1999.
- [20] S. Hodžić and E. Pasalic, “Generalized bent functions – Some general construction methods and related necessary and sufficient conditions,” *Cryptogr. Commun.*, vol. 7, pp. 469–483, 2015.
- [21] N. Jefferies, “Sporadic partitions of binomial coefficients,” *Elec. Letters*, vol. 27, no. 15, pp. 134–136, 1991.
- [22] J.B. Kam and G.I. Davika, “Structured Design of Substitution-Permutation Encryption Networks,” *IEEE Trans. on Computers*, vol. 28, no. 10, p. 747, 1979.
- [23] P.V. Kumar, R.A. Scholtz, and L.R. Welch, “Generalized bent functions and their properties,” *J. Combin theory*, ser. A, vol. 40, pp. 90–107, 1985.
- [24] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd ed., Encyclopedia Math. Appl., vol. 20, Cambridge, UK, Cambridge Univ. Press, 1997.
- [25] P. Lisonek and H.Y. Lu, “Bent functions on partial spreads,” *Des. Codes Cryptogr.*, vol. 73, pp. 209–216, 2014.
- [26] F.J. MacWilliams and N.J.A Sloane, *The Theory of Error-Correcting Codes*, Amsterdam, Netherlands, North-Holland, 1977.
- [27] E.J. Ionascu, T. Martinsen, and P. Stănică, “Bisecting binomial coefficients,” *Discrete Appl. Math.*, vol. 227, pp. 70–83, 2017.
- [28] T. Martinsen, W. Meidl, and P. Stănică, “Generalized bent functions and their Gray images,” in *Arithmetic of Finite Fields, 6th Int. Workop, WAIFI 2016*, ser. LNCS, vol. 10064, Berlin, Springer, 2016, pp. 160–173.
- [29] T. Martinsen, W. Meidl, and P. Stănică, “Partial spread and vectorial generalized bent functions,” *Des. Codes Cryptogr.*, vol. 85, pp. 1–13, 2017.
- [30] W. Meier and O. Stafflebach, “Nonlinear criteria for cryptographic functions,” in *Adv. in Crypt.-Eurocrypt ’89*, ser. LNCS, vol. 434, Berlin, Springer, 1990, pp. 549–562.
- [31] K. Nyberg, “Perfect nonlinear S-boxes,” in *Advances in cryptology, EUROCRYPT ’91*, ser. LNCS, vol. 547, Berlin, Springer, 1991, pp. 378–386.
- [32] M.G. Parker and A. Pott, “On Boolean functions which are bent and negabent,” in *Sequences, subsequences, and consequences*, ser. LNCS, vol. 4893, Berlin, Springer, 2007, pp. 9–23.

- [33] J.P. Pieprzyk and C.X. Qu, “Fast hashing and rotation-symmetric functions,” *J Universal Computer Science*, vol. 5, pp. 20–31, 1999.
- [34] O.S. Rothaus, “On bent functions,” *J. Combin. Theory*, ser. A, vol. 20, pp. 300–305, 1976.
- [35] J. Rouse, “Explicit bounds for sums of squares,” *Math. Res. Lett.*, vol. 19, no. 2, pp. 359–376, 2012.
- [36] P. Sarkar and S. Maitra, “Cross–Correlation Analysis of Cryptographically Useful Boolean Functions and S-Boxes,” *Theory Comput. Systems*, vol. 35, pp. 39–57, 2002.
- [37] K.U. Schmidt, “Quaternary constant-amplitude codes for multicode CDMA,” *IEEE Trans. Inform. Theory*, vol. 55, no. 4, pp. 1824–1832, 2009.
- [38] K.U. Schmidt, “ \mathbb{Z}_4 -valued quadratic forms and quaternary sequence families,” *IEEE Trans. Inform. Theory*, vol. 55, no. 12, pp. 5803–5810, 2009.
- [39] C.E. Shannon, “Communication theory of secrecy systems,” *Bell Systems Technical Journal*, vol. 28, pp. 656–715, 1949.
- [40] T. Siegenthaler, “Correlation immunity of nonlinear combining functions for cryptographic applications,” *IEEE Trans. Inform. Theory*, vol. 30, no. 1, pp. 776–780, 1984.
- [41] N.J.A Sloane, Directory of Orthogonal Arrays. [online]. Available: <http://neilsloane.com/oadir/>. Accessed Jun. 10, 2017.
- [42] P. Solé and N. Tokareva, “Connections between Quaternary and Binary Bent Functions,” *Prikl. Diskr. Mat.*, vol. 1, pp. 16–18, 2009.
- [43] P. Stănică, S. Gangopadhyay, A. Chaturvedi, A.K. Gangopadhyay, and S. Maitra, “Investigations on bent and negabent functions via the nega-Hadamard transform,” *IEEE Trans. Inform. Theory*, vol. 58, no. 6, pp. 4064–4072, 2012.
- [44] P. Stănică, T. Martinsen, S. Gangopadhyay, and B.K. Singh, “Bent and generalized bent Boolean functions,” *Des. Codes Cryptogr.*, vol. 69, pp. 77–94, 2013.
- [45] P. Stănică and S. Maitra, “A constructive count of rotation symmetric functions,” *Inform Process Lett.*, vol. 88, pp. 299–304, 2003.
- [46] P. Stănică and S. Maitra, “Rotation symmetric Boolean functions-count and cryptographic properties,” *Discrete Appl. Math.*, vol. 156, pp. 1567–80, 2008.
- [47] W. Su, A. Pott, and X. Tang, “Characterization of negabent functions and construction of bent-negabent functions with maximum algebraic degree,” *IEEE Trans. Inform. Theory*, vol. 59, no. 6, pp. 3387–3395, 2013.
- [48] N. Tokareva, “Generalizations of bent functions: a survey of publications,” *J. Appl. Ind. Math.*, vol. 5, no. 1, pp. 110–129, 2011.

- [49] N. Tokareva, *Bent Functions, Results and Applications to Cryptography*, San Diego, CA, Academic Press, 2015.
- [50] A.F. Webster and S.E. Tavares, “On the design of S-boxes,” in *Adv. in crypt.-Crypto '85*, ser. LNCS, vol. 218, Berlin, Springer, 1986, pp. 523–534.
- [51] C. Wu and E. Dawson, “Construction of correlation immune Boolean functions,” *Australasian J. Combinatorics*, vol. 21, pp. 141–166, 2000.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California